Using M68HC16 Digital Signal Processing To Build An Audio Frequency Analyzer

By Mark Glenewinkel

Freescale Semiconductor, Inc.

INTRODUCTION

This application note demonstrates the use of a microcontroller unit (MCU) with integrated DSP capabilities. The MC68HC16Z1 is a high performance 16-bit MCU that includes on-chip peripheral modules and a CPU module (CPU16). The CPU16 instruction set simplifies the use of digital signal processing algorithms, and makes it easy to implement low-bandwidth filter and control-oriented applications.

OBJECTIVES

The goal of this application note is for an engineer to learn the MC68HC16Z1 well enough to design and build an audio frequency analyzer (AFA). The following intermediate objectives have been defined to help reach this goal.

- Learning the CPU16 instruction set
- Becoming familiar with MC68HC16Z1 modules
- Learning basic MCU I/O hardware and software
- Understanding DSP system concepts with the frequency analyzer
- Understanding and implementing common DSP algorithms with an MCU

This is a tutorial design project that follows a hands-on approach to using DSP. It provides concrete hardware/software applications that are used to understand and design an MCU-based system utilizing DSP al-

© Freescale Semiconductor, Inc., 2004. All rights reserved.

EQUIPMENT REQUIRED

The following items are needed to build and test the audio frequency analyzer (AFA).

1. An IBM PC compatible computer with a parallel printer port
2. The M68HC16Z1EVB
3. A prototyping or wire-wrap board
4. One straight DB25 cable, male on one end, female on the other
5. A 5 volt power supply
6. An audio sound source, preferably a CD player
7. Two Y-connectors to split the stereo sound source with audio cables
8. A sinusoidal waveform generator, optional
9. Oscilloscope for debugging, optional

All of the components needed to build the AFA are shown in **Figure 4** and **Figure 5**, the AFA schematics.

THE AUDIO FREQUENCY ANALYZER

Spectral analysis is a method of determining the specific frequency content of a signal and the energy levels of these frequencies. This information is processed by either Fourier Transform methods or by specific filtering of the signal. The information is tabulated for more analysis or displayed in a visual format.

One example of spectral analysis is found in oil exploration. An engineer sends a known signal into the earth and then calculates the frequency content of the reflected signal. This is a classic input/output black box. The transfer function of the black box (the earth in this case) yields clues to the structure beneath the surface. Different frequency responses correspond to different types of rock. With spectral analysis, the engineer can decide whether it is feasible to drill.

This project focuses on the frequency analysis of an audio signal. A frequency analyzer is often used in audio systems and recording studios. It filters out energy levels of specific audio frequencies and displays them to indicate the frequency content of the audio signal. Audio frequency analyzers are also used in conjunction with equalizers to help the user define and shape the spectral characteristics of a sound source.

Figure 1 is a generic system diagram of a frequency analyzer based on bandpass filters. The input signal is split and sent to all the filters. The filters pass only specific frequency components of the input signal. After filtration, the strength of each passed signal is analyzed, and the amount of energy in each band is repre-

Freescale Semiconductor, Inc.

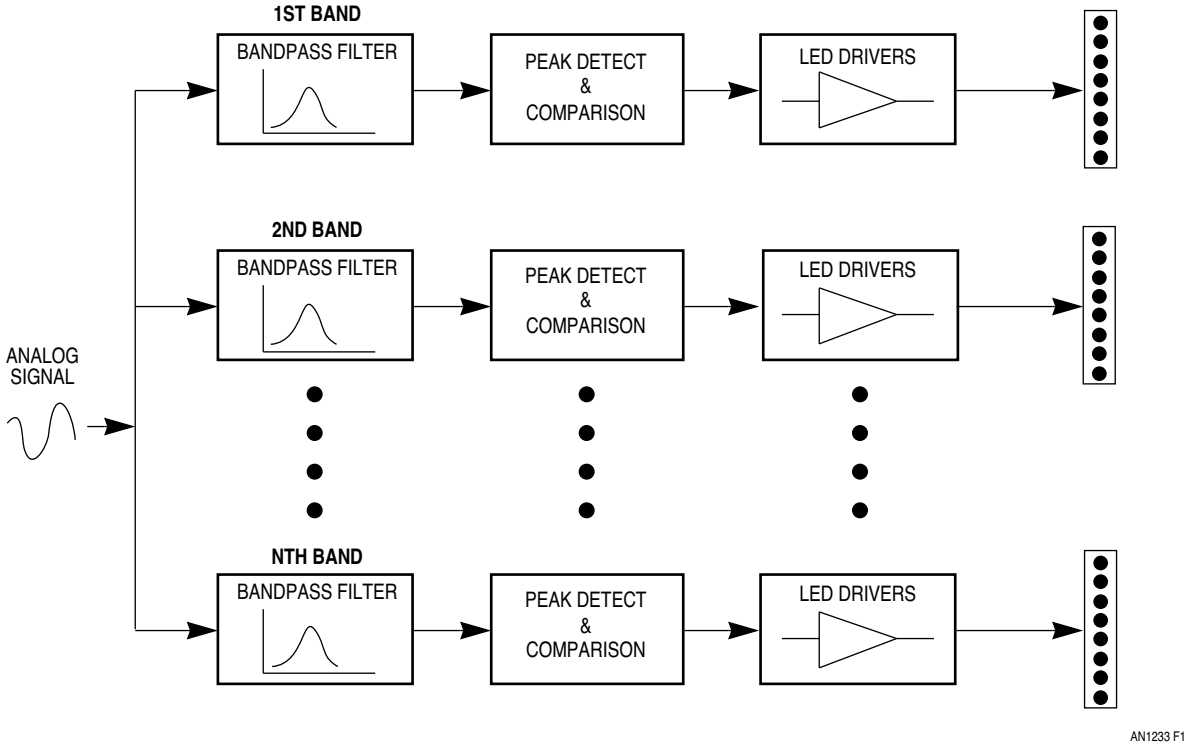
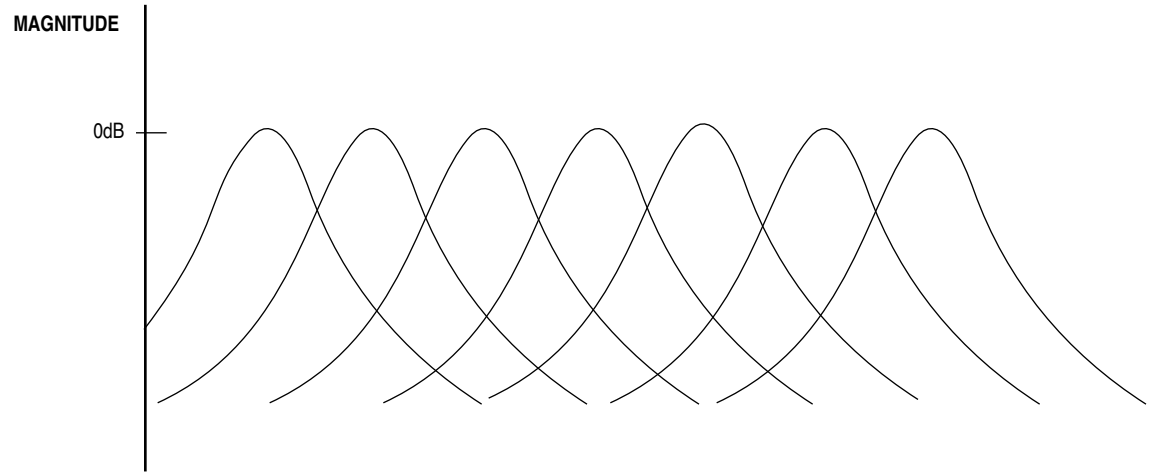


Figure 1 Frequency Analyzer System Diagram



Freescale Semiconductor, Inc.

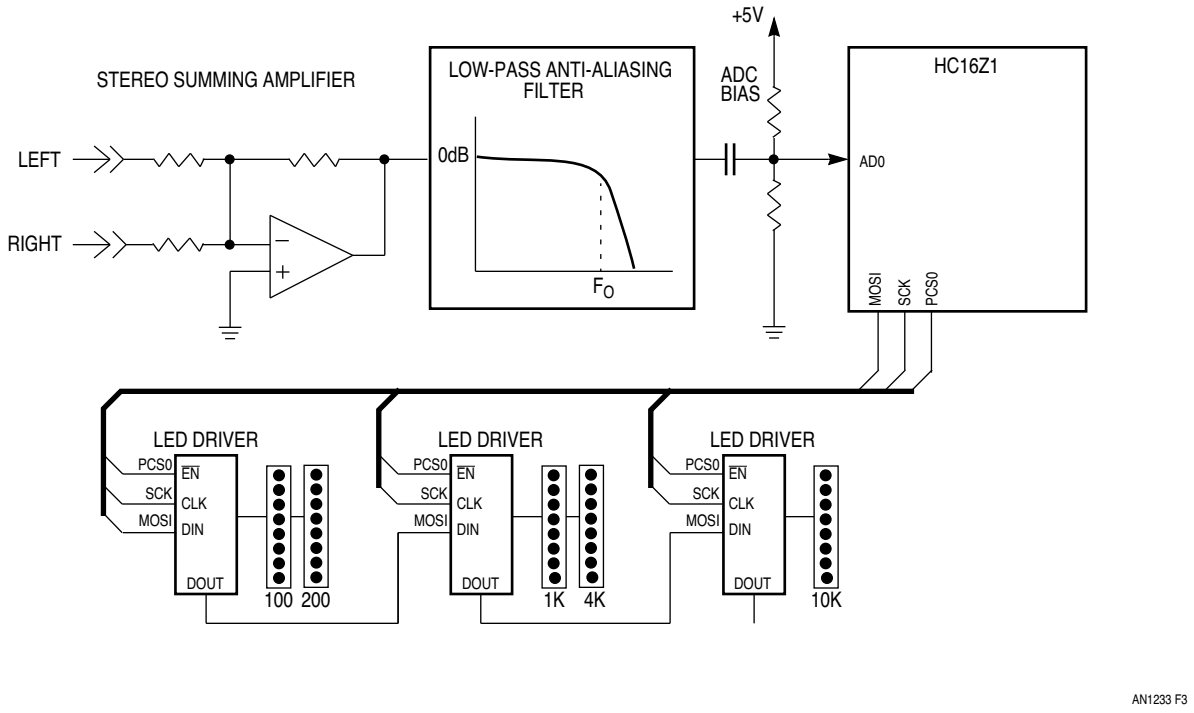
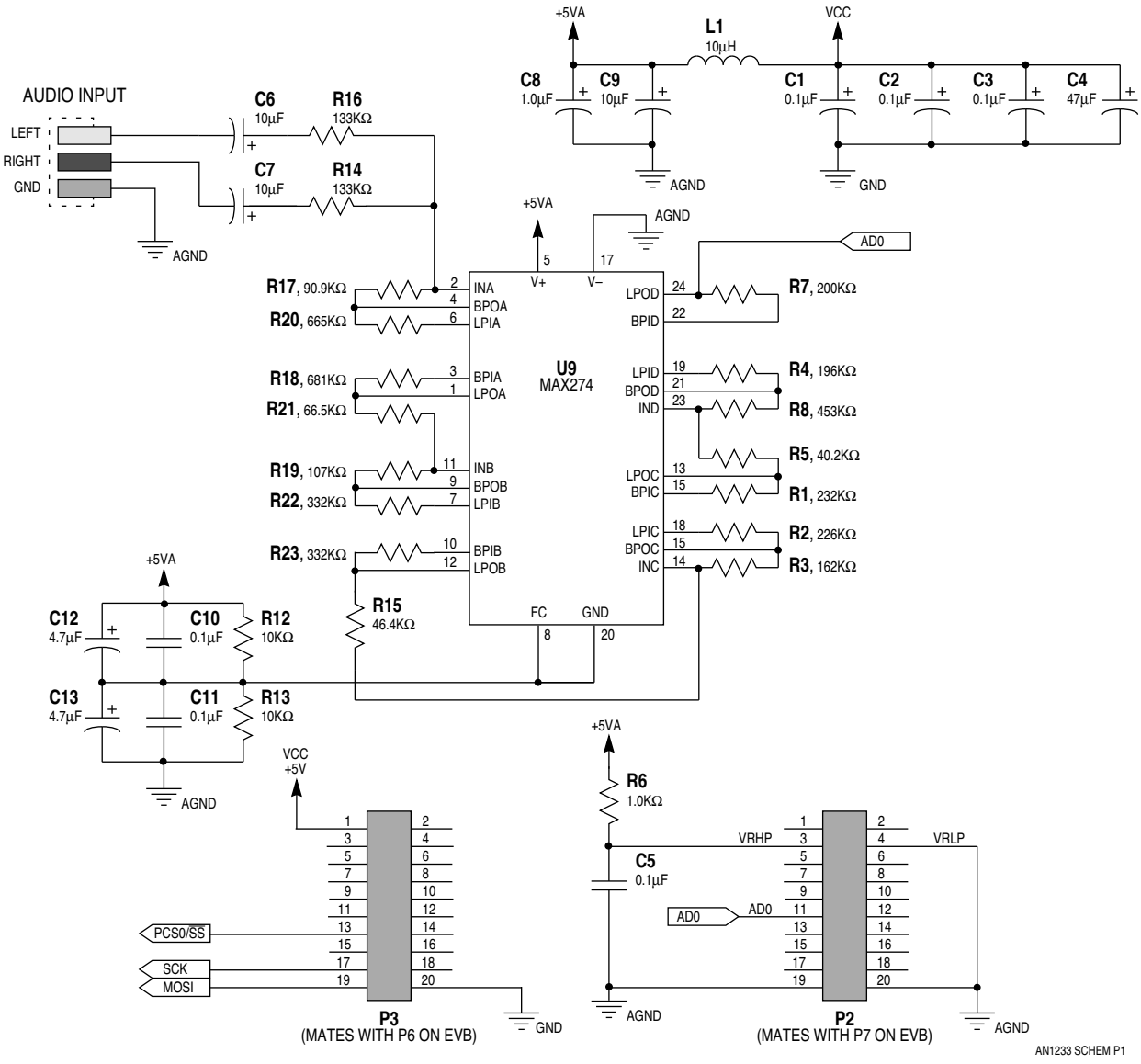


Figure 3 Audio Frequency Analyzer System Diagram

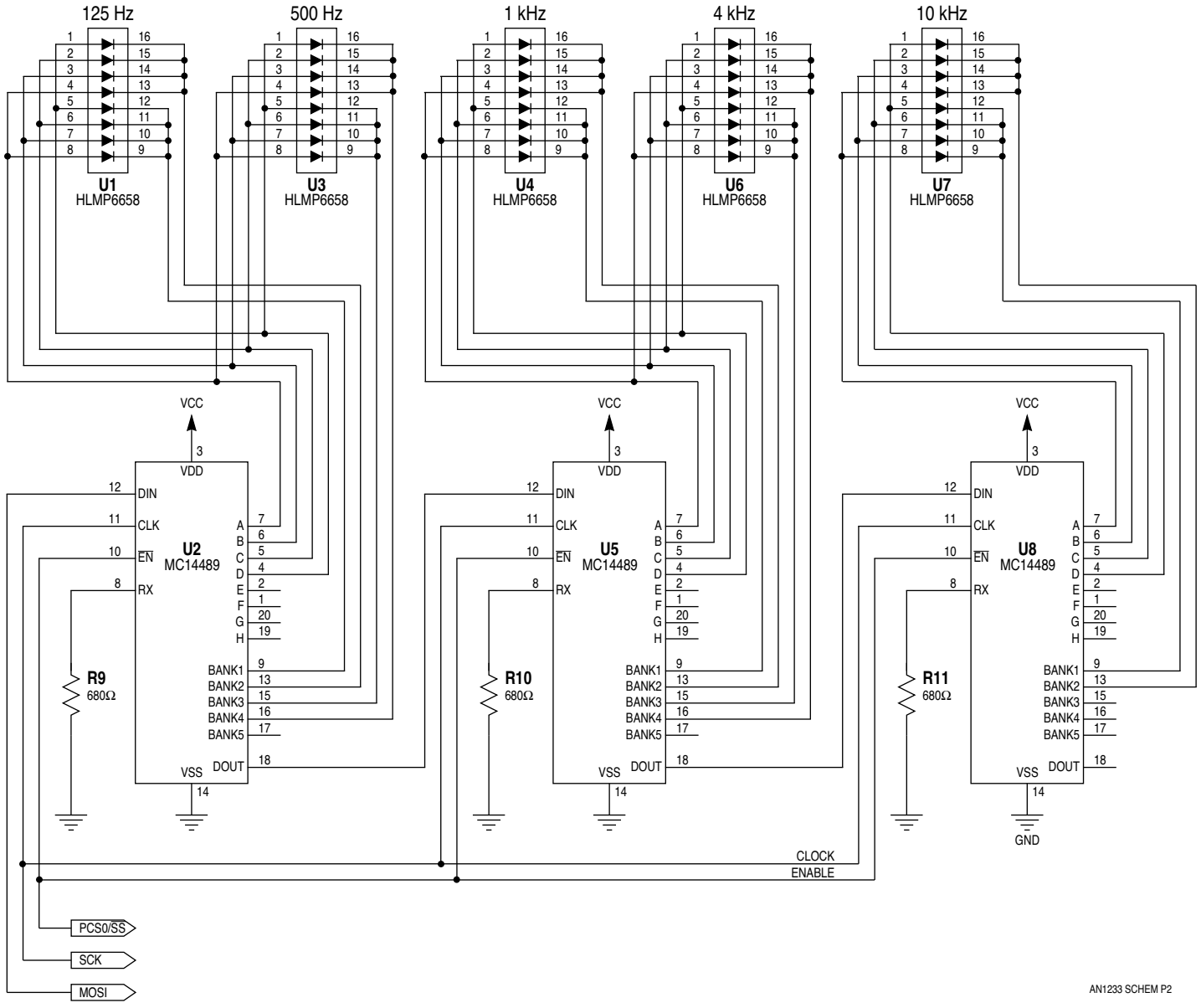
AFA Hardware

Familiarity with the AFA hardware helps to understand the code used to implement the analyzer. **Figure 4** is a schematic of the analog front end of the AFA, and **Figure 5** is a schematic of the display logic.

Freescale Semiconductor, Inc.



Freescale Semiconductor, Inc.

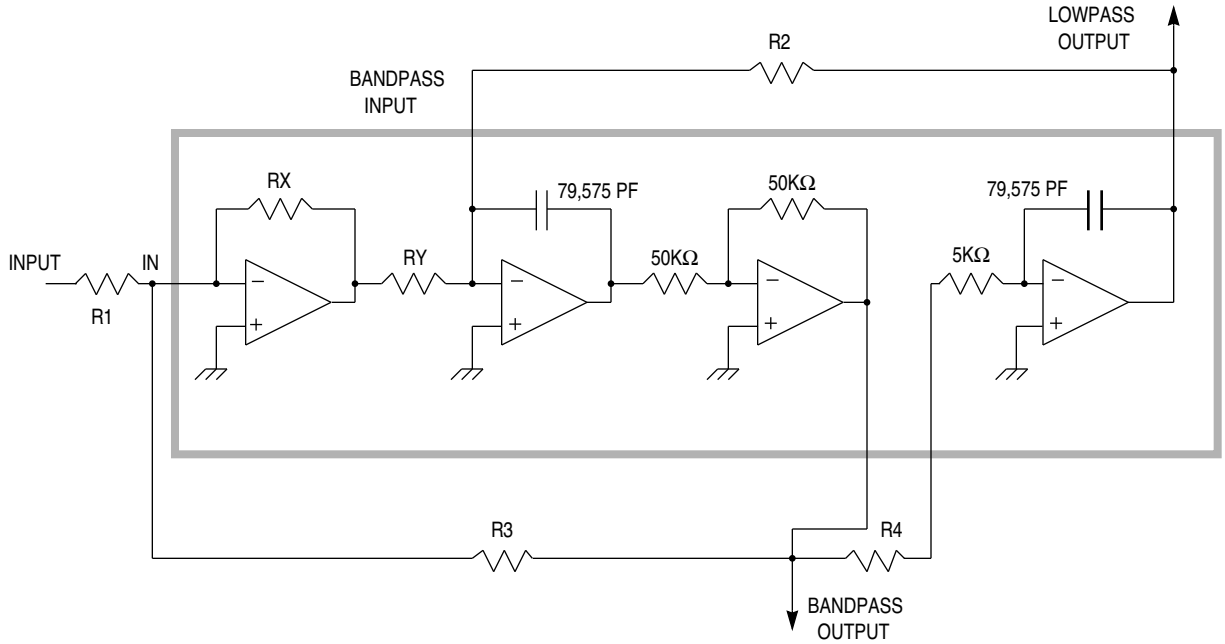


Freescale Semiconductor, Inc.

The Analog Front End

The analog front end contains all of the circuitry to condition the signal for analog-to-digital conversion and subsequent digital signal processing. It consists of the summing circuitry for the stereo signal, the anti-aliasing filter, and the biasing circuitry for the ADC. A MAX274 low-pass filter chip, manufactured by the Maxim Corporation of Sunnyvale, California, is used to implement all of these functions.

The MAX274 is an eighth order, programmable, continuous-time active filter. The chip consists of four independent cascadable second-order filter sections. Each filter section can implement any all-pole bandpass or lowpass filter, characterized as a Butterworth, Bessel, or Chebyshev response. Each second-order section is programmable with four external resistors. A second-order section is illustrated in **Figure 6**. Maxim provides an evaluation board and a software package that calculates resistor values from response specifications input by the user. This makes the MAX274 very flexible and easy to use when implementing high-order anti-aliasing filters.



Freescale Semiconductor, Inc.

Anti-Aliasing Filter

When a signal of a given frequency is sampled at too low a rate, it appears as a totally different lower frequency at the output of the sampler. This phenomenon is referred to as aliasing. Aliasing occurs at a point called the folding frequency, which is one-half the sampling frequency. In order for the frequency analyzer to be accurate, sampling frequency must therefore be at least two times the highest frequency component to be sampled. The ideal solution to this problem is to raise the sampling rate as high as possible, but real-world designs generally have a fixed upper limit on sampling frequency. The most practical solution is to attenuate high frequency components of the input signal so that aliasing does not occur. The anti-aliasing filter correctly attenuates the high frequency components of the signal, so that they are not present within the sample bandwidth.

The AFA has a 25-kHz sampling frequency (Fs), and a processing bandwidth of 10 kHz. If no filter is used, signal components with a frequency higher than 12.5 kHz alias at lower frequencies, and the digitized samples represent invalid information. **Figure 7** shows these relationships. Fs/2 is the folding frequency, 12.5 kHz. Frequencies that will not alias with a 25 kHz sampling frequency are to the left of Fs/2, while frequencies that will alias are to the right of Fs/2.

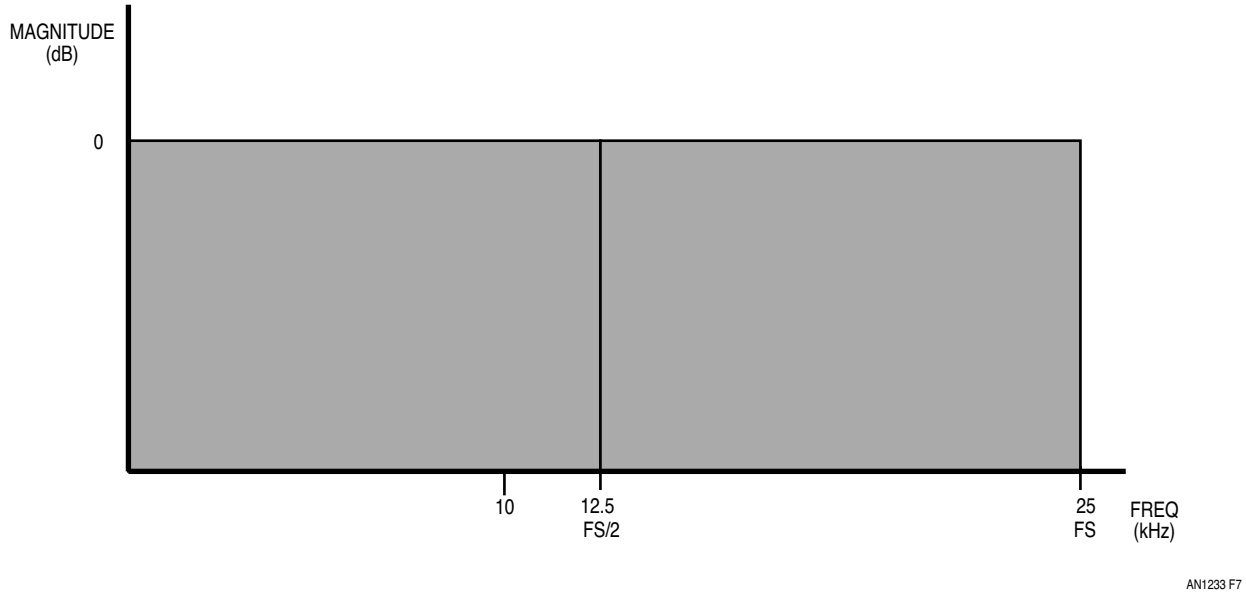


Figure 7 AFA Aliasing Without Filter

Freescale Semiconductor, Inc.

Voltage Attenuation (dB) = 20 ∗ log [1/(2^{ADCres})]

Where:

ADCres = A/D converter resolution

System bandwidth is 10 kHz, and at a 25 kHz sampling frequency, components above 12.5 kHz will alias. Therefore, the signal must be attenuated 48 dB to eliminate all aliasing components. Accordingly, the filter must have a minimum drop-off slope of 96 dB per octave. To insure that this requirement is met, a roll-off of 100 dB per octave is used. Using these values with the MAX274 design software, resistor values for an eighth order 0.5 dB passband ripple Chebyshev filter were obtained. Lower passband ripple was sacrificed to gain steeper roll-off. The anti-aliasing filter response programmed into the MAX274 is shown in **Figure 8**.

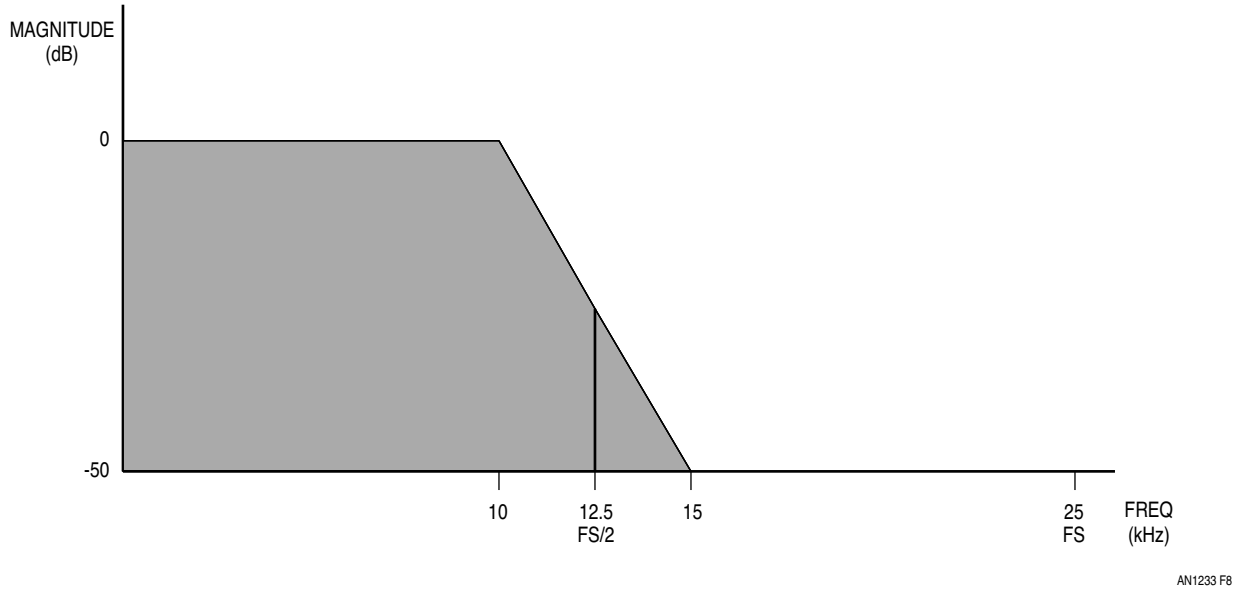
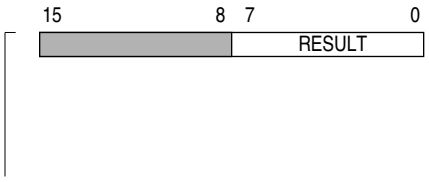


Figure 8 AFA Anti-Aliasing Filter Roll-Off

ADC Input Biasing

The MC68HC16Z1 ADC module can convert analog data into six different digital representations. Digital data can have 8-bit or 10-bit resolution, can be signed or unsigned, and can be left or right justified. These formats are shown in **Figure 9**.



Freescale Semiconductor, Inc.

Figure 10 shows hexadecimal representations of signed and unsigned ADC data. For 8-bit conversions, there are 256 possible values. Unsigned formats assume the zero voltage point is at the low ADC reference voltage, with 256 steps from low to high reference. Signed formats assume that the zero voltage point is halfway between the low and high ADC reference voltages. The most significant bit indicates a positive or negative value — 128 values represent positive voltages, and 128 two's-complement values represent negative voltages ($00 represents the midpoint, and $FF represents midpoint minus one count).

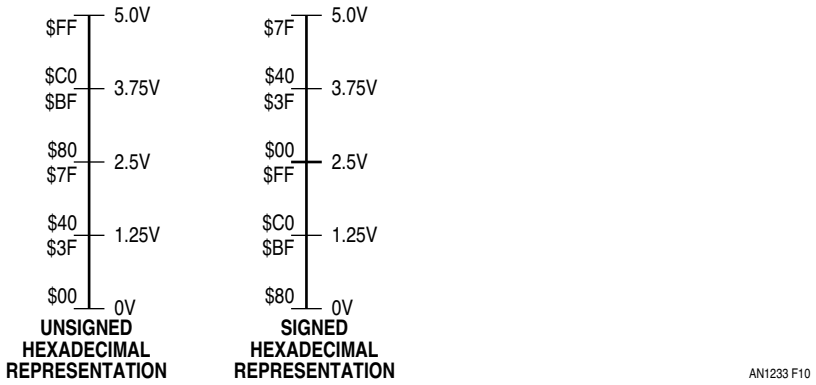


Figure 10 Hexadecimal Representation of 8-Bit ADC Data

The AFA uses signed, 8-bit, left justified ADC data. The analog signal must be biased at 2.5 vdc, centered between the 0 vdc and 5 vdc ADC reference voltages, in order to use this representation. The MAX274 is used to bias the signal.

The MAX274 requires two power connections. Biasing circuitry consists of a voltage divider (R12, R13) and decoupling capacitors (C10 – C13) connected to one of the MAX274 supplies. The V– pin is connected to analog ground. The V+ pin is connected to the 5 volt supply. The GND pin is connected to 2.5 volts. This splits the supply and causes the analog signal to have a 2.5 volt DC offset. The signal is buffered by an op amp driver and is sent directly to the ADC module pins from the MAX274. The ADC can now properly sample the signal.

The Digital Back End

The digital back end shown in **Figure 5** contains all of the circuitry required to output digitally processed information to the LED array. When digital signal processing is complete, encoded energy levels for each band are loaded into QSPI transmit RAM, then the QSPI is activated, and the data is transmitted serially to the MC14489 LED drivers.

Freescale Semiconductor, Inc.

Three MC14489 drivers are used in the AFA. There are five 8-bit LED arrays. Two of the MC14489 chips control four banks of four diodes each, and one controls two banks of four diodes each. Drive current for diodes in each bank is supplied by pins A, B, C, and D of the MC14489. The cathodes of each bank of diodes are tied together and a bank-select pin sinks the current for that bank. Please refer to the MC14489 Data Sheet for more information.

The M68HC16Z1 EVB and Development Environment

The M68HC16Z1 Evaluation Board provides the capability to test and debug the audio frequency analyzer. **Table 1** shows development software supplied with the EVB.

Table 1 Development Software

MASM16.EXE
MASM.EXE
HEX.EXE
MASM16.HLP
EVB16.EXE

MASM16 software is used to edit and assemble code, and EVB16 software is used to download code to the EVB and run it. EVB16 software also has debug capabilities such as trace and breakpoint. Please refer to the *M68HC16Z1EVB User's Manual* for a list of debug features.

Assembling the Development Environment

Assembling the development system with the AFA is simple. Hook up the system as shown in **Figure 11**. The AFA project board connects to the M68HC16Z1EVB via P7 and P6. Use the DB25 cable to connect the parallel port of the PC to the parallel port connector of the EVB. After connecting the 5 volt power supply to the M68HC16Z1EVB, connect the audio signal source. A CD player is the recommended source for a high quality output. Split the audio source outputs so that both the AFA board and the speakers receive the signals (audio splitters can be found at most stereo and electronics stores).

Freescale Semiconductor, Inc.

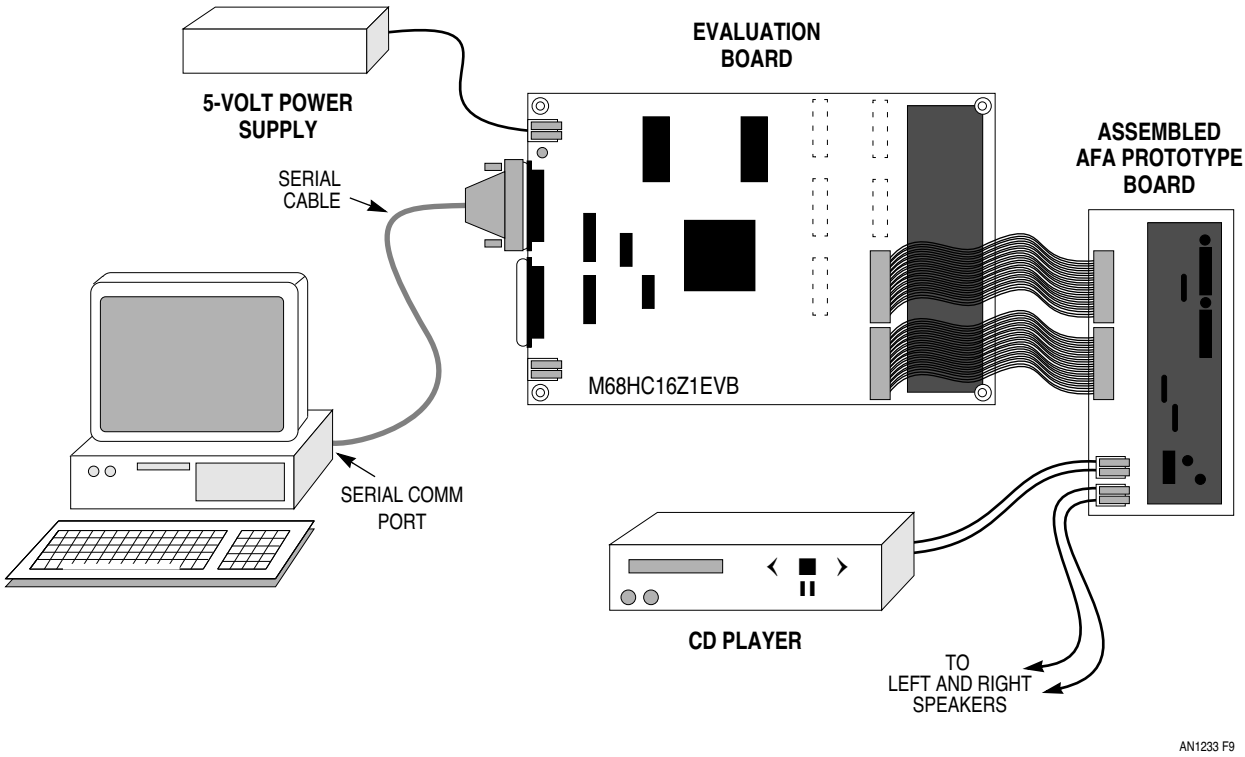


Figure 11 AFA Development System Setup

Freescale Semiconductor, Inc.

AFA Software

Even though hardware is required to build the AFA, software running on the CPU16 performs most of the actual work. Five tutorial programs must be integrated to complete the project. Each program demonstrates specific functions of the AFA, and each is discussed in a separate section. Since this is a DSP project/tutorial, discussion focuses on signal-processing tasks. Each of the tutorial programs must be modified in order to complete the AFA. The software steps to the AFA design are listed below.

1. Acquisition of data
2. QSPI to MC14489 interface
3. Periodic interrupt timer routine
4. Peak detector
5. 1-kHz bandpass filter routine
6. 5-band audio frequency analyzer

AFA software is listed in **Table 2**. Each of the first six programs in the table corresponds to one of the software steps listed above. In order to organize and streamline the project, each program has been designed according to a standard template for the M68HC16Z1EVB. **Figure 12** shows the template.

Table 2 AFA Project Software

ADC.ASM
QSPI_LED.ASM
INT_TEST.ASM
PEAK.ASM
1K_FLTR.ASM
5BAND_SA.ASM
EQUATES.ASM
ORG00000.ASM
INITSYS.ASM
INITRAM.ASM
OUTVAL1.ASM
OUTVAL2.ASM

OUTVAL1.ASM and OUTVAL2.ASM are lookup tables for the LED display routines. They contain values that correspond to the number of LEDs needed to reflect a given peak value.

Freescale Semiconductor, Inc.

```
*
*       MOTOROLA, INC.
*       Advanced MCU Division
*       Austin, Texas
*
*       Title: HC16 SOFTWARE TEMPLATE
*
*       File Name: TEMPLATE.ASM
*
*       Description: This program provides a template for all
*                      designers to use with the HC16Z1
*                    An equate table is given.
*                    The reset vector is initialized.
*                    The CPU and RAM are also initialized.
*                    The user can put his code in the 'user area'
*                      block of this template
*
*       History: 06/05/91 Created.
*                10/02/91 Modified comments.
*
*       Note: This program is written for the M68HC16Z1EVB.
*******************************************************************************
        INCLUDE 'EQUATES.ASM'  ;table of EQUates for common register addr
        INCLUDE 'ORG00000.ASM' ;initialize reset vector

        ORG     $0200          ;start program after interrupt vectors

*****   Initialization Routines *****
        INCLUDE 'INITSYS.ASM'  ;initially set EK=F, XK=0, YK=0, ZK=0
                               ;set sys clock at 16.78 MHz, disable COP
        INCLUDE 'INITRAM.ASM'  ;initialize and turn on SRAM
                               ;set stack (SK=1, SP=03FE)


*****   Start of user program area  *****
```

Figure 12 AFA Software Template

Freescale Semiconductor, Inc.

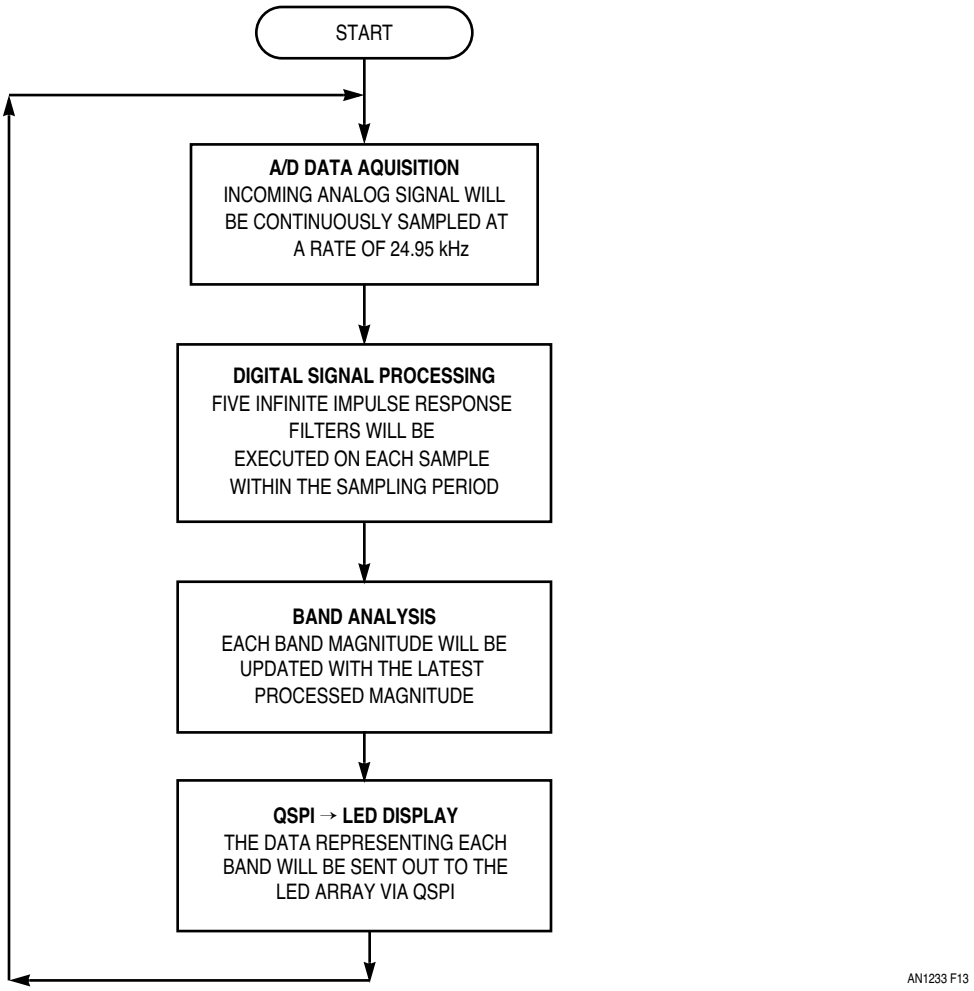

START
A/D DATA AQUISITION
INCOMING ANALOG SIGNAL WILL BE CONTINUOUSLY SAMPLED AT A RATE OF 24.95 kHz
DIGITAL SIGNAL PROCESSING
FIVE INFINITE IMPULSE RESPONSE FILTERS WILL BE EXECUTED ON EACH SAMPLE WITHIN THE SAMPLING PERIOD
BAND ANALYSIS
EACH BAND MAGNITUDE WILL BE UPDATED WITH THE LATEST PROCESSED MAGNITUDE
QSPI → LED DISPLAY
THE DATA REPRESENTING EACH BAND WILL BE SENT OUT TO THE LED ARRAY VIA QSPI
AN1233 F13

Figure 13 AFA System Software Flowchart

All processing must be completed within one period of the 24.95-kHz sampling frequency. As shown below,

Freescale Semiconductor, Inc.

Figure 14 shows the relationship between sampling periods and real-time digital signal processing. All calculations and internal/external housekeeping must be taken care of within the given sample period.

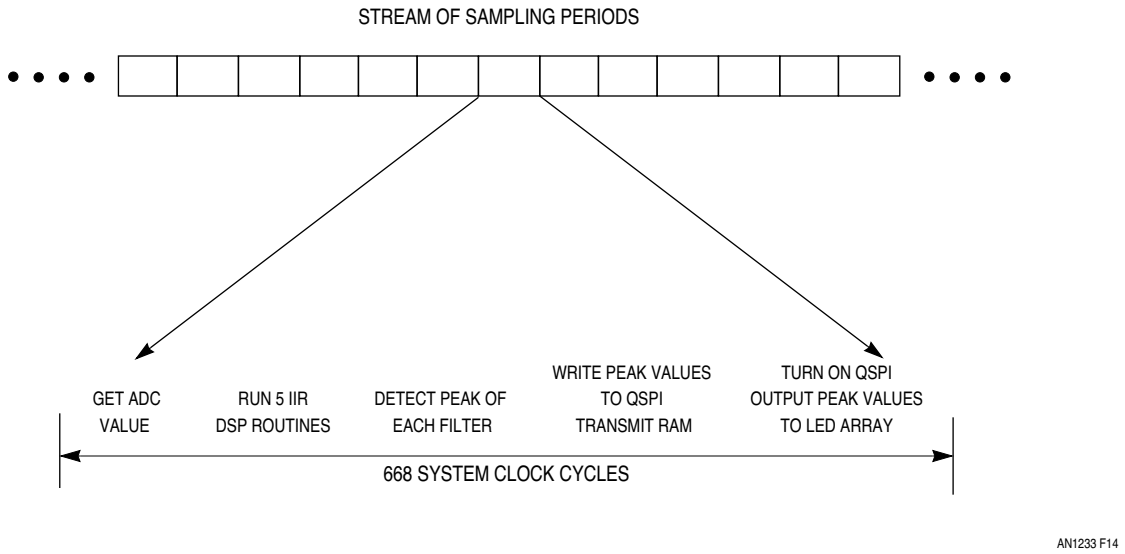

STREAM OF SAMPLING PERIODS
GET ADC VALUE
RUN 5 IIR DSP ROUTINES
DETECT PEAK OF EACH FILTER
WRITE PEAK VALUES TO QSPI TRANSMIT RAM
TURN ON QSPI OUTPUT PEAK VALUES TO LED ARRAY
668 SYSTEM CLOCK CYCLES
AN1233 F14

Figure 14 AFA Sampling Period

Software Design Implementation

The following sections examine AFA software in detail. For each of the programs, there is a discussion of design and implementation, a code listing, and appropriate flow charts. In the interest of brevity, the standard template headers have been omitted from the listings, and redundant portions of flowcharts are reproduced only once.

Analog-to-Digital Data Acquisition (ADC.ASM)

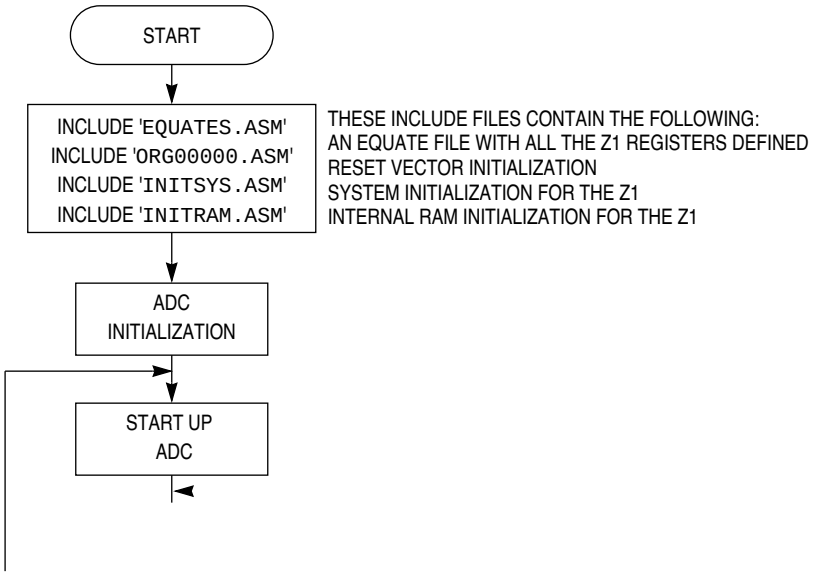
In order to perform digital signal processing, a digital representation of the analog signal must be available. The MC68HC16Z1 contains a programmable ADC module. The ADC has a number of automatic conversion modes. Only four registers are needed to control the ADC. Refer to the *ADC Reference Manual* (ADCRM/AD) for more detailed information.

ADC.ASM initializes the ADC module, then goes into a continuous loop, repeating the programmed con-

Freescale Semiconductor, Inc.

ADC.ASM Code listing

```
        INCLUDE 'EQUATES.ASM'   ;table of EQUates for common register addr
        INCLUDE 'ORG00000.ASM'  ;initialize reset vector

        ORG     $0200

*****  Initialization Routines  *****

        INCLUDE 'INITSYS.ASM'   ;initially set EK=F, XK=0, YK=0, ZK=0
                                ;set sys clock at 16.78 MHz, disable COP
        INCLUDE 'INITRAM.ASM'   ;initialize and turn on SRAM
                                ;set stack (SK=1, SP=03FE)

ORG     $0200
*****   ADC Initialization      *****
        LDD     #$0000
        STD     ADCMCR          ;turn on ADC
        LDD     #$0003
        STD     ADCTL0          ;8-bit, set sample period

*****   ADC Start               *****
LOOP    LDD     #$0000
        STD     ADCTL1          ;single 4 conversion, single channel, AD0
                                ;writing to the ADCTL1 reg starts conversion

        LDAA    #$80
SCFSET  BITA    ADSTAT          ;check for the Sequence Complete Flag
        BEQ     SCFSET          ;complete?, if not check again
        BRA     LOOP            ;go get another sample
```



Freescale Semiconductor, Inc.

QSPI TO MC14489 Interface (QSPI_LED.ASM)

This program illustrates QSPI serial timing and data format, which must be understood in order to program the QSPI to talk to the MC14489. The *QSM Reference Manual* (QSMRM/AD) and the MC14489 data sheet are needed to understand the code.

QSPI_LED.ASM initializes the QSPI module and the three MC14489 drivers to handle 40 LEDs. After this it updates the LED array by writing to the MC14489 display registers, then gives control back to the EVB16 development software. Values being sent to the array may be changed either by modifying the memory locations that hold the transmitted data or by reassembling the lines that load these memory locations. **Figure 16** is a flowchart of **QSPI_LED.ASM**.

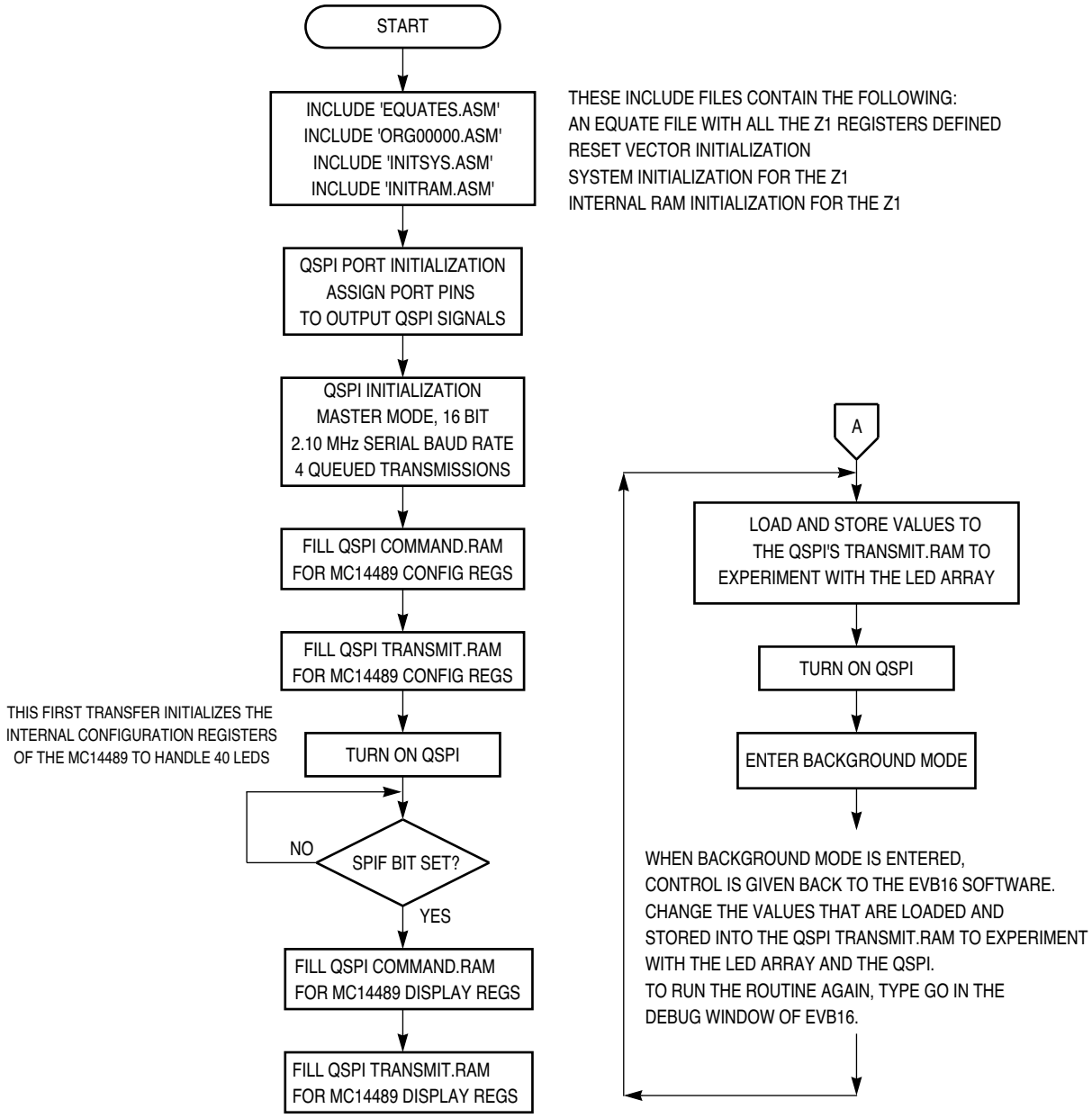
QSPI_LED.ASM Code Listing

```
        INCLUDE 'EQUATES.ASM'   ;table of EQUates for common register addr
        INCLUDE 'ORG00000.ASM'  ;initialize reset vector

        ORG     $0200

*****  Initialization Routines  *****

        INCLUDE 'INITSYS.ASM'   ;initially set EK=F, XK=0, YK=0, ZK=0
                                ;set sys clock at 16.78 MHz, disable COP
        INCLUDE 'INITRAM.ASM'   ;initialize and turn on SRAM
                                ;set stack (SK=1, SP=03FE)

*****   QSPI Initialization     *****
        LDAA    #$08
        STAA    QPDR            ;output pcs0/ss* to 0 when asserted
        LDAA    #$0F
        STAA    QPAR            ;assign QSM port pins to qspi module
        LDAA    #$FE
        STAA    QDDR            ;assign all QSM pins as outputs except miso

        LDD     #$8004          ;mstr, womq=cpol=cpha=0
        STD     SPCR0           ;16 bits, 2.10MHz serial baud rate
        LDD     #$0300          ;no interrupt generated, no wrap mode
        STD     SPCR2           ;newqp=0, endqp=3, queued for 4 trans

*****   Fill QSPI Command.ram to write the config registers of the 14489
        LDAA    #$C0
        STAA    CR0             ;cont=1, bitse=1, pcs0=0, no delays needed
        STAA    CR1
        STAA    CR2
        LDAA    #$40
        STAA    CR3             ;cont=0, bitse=1, pcs0=0, no delays needed

*****   Fill QSPI Transmit.ram to write the config registers of the 14489
        LDAA    #$3F
        STD     TR0+1           ;store $3F to tran.ram registers
```

Freescale Semiconductor, Inc.

```
*****   Fill QSPI Transmit.ram for display registers of the 14489
*****   The beginning LED values will be $00, all of the LEDs will be off
        LDD     #$8000          ;TR0 = $8000
        STD     TR0             ;TR1 = $0080
        STAA    TR3+1           ;TR2 = $0000
        LDD     #$0080          ;TR3 = $XX80
        STD     TR1             ;TR4 = $0000
        CLRD
        STD     TR2
        STD     TR4

        LDD     #$0400          ;display registers need 5 transmissions
        STD     SPCR2           ;newqp=0, endqp=4

*****   Load up the various LED bands for experimentation
T125    LDAA    #$0F
        STAA    TR4+1           ;125 Hz band
T500    LDAA    #$3F
        STAA    TR4             ;500 Hz band
T1K     LDAA    #$FF
        STAA    TR2+1           ;1k Hz band
T4K     LDAA    #$3F
        STAA    TR2             ;4k Hz band
T10K    LDAA    #$03
        STAA    TR1             ;10k Hz band

        LDD     #$8404          ;load up d
        STD     SPCR1           ;turn on QSPI

        BGND                    ;go back to EVB16 software
                                ;reassemble code for T125 to T10K
                                ;experiment with different values

        BRA     T125            ;branch back to TR125 line
```

Freescale Semiconductor, Inc.


START
INCLUDE 'EQUATES.ASM'
INCLUDE 'ORG00000.ASM'
INCLUDE 'INITSYS.ASM'
INCLUDE 'INITRAM.ASM'
THESE INCLUDE FILES CONTAIN THE FOLLOWING:
AN EQUATE FILE WITH ALL THE Z1 REGISTERS DEFINED
RESET VECTOR INITIALIZATION
SYSTEM INITIALIZATION FOR THE Z1
INTERNAL RAM INITIALIZATION FOR THE Z1
QSPI PORT INITIALIZATION
ASSIGN PORT PINS
TO OUTPUT QSPI SIGNALS
QSPI INITIALIZATION
MASTER MODE, 16 BIT
2.10 MHz SERIAL BAUD RATE
4 QUEUED TRANSMISSIONS
FILL QSPI COMMAND.RAM
FOR MC14489 CONFIG REGS
FILL QSPI TRANSMIT.RAM
FOR MC14489 CONFIG REGS
THIS FIRST TRANSFER INITIALIZES THE
INTERNAL CONFIGURATION REGISTERS
OF THE MC14489 TO HANDLE 40 LEDS
TURN ON QSPI
NO
SPIF BIT SET?
YES
FILL QSPI COMMAND.RAM
FOR MC14489 DISPLAY REGS
FILL QSPI TRANSMIT.RAM
FOR MC14489 DISPLAY REGS
A
LOAD AND STORE VALUES TO
THE QSPI'S TRANSMIT.RAM TO
EXPERIMENT WITH THE LED ARRAY
TURN ON QSPI
ENTER BACKGROUND MODE
WHEN BACKGROUND MODE IS ENTERED,
CONTROL IS GIVEN BACK TO THE EVB16 SOFTWARE.
CHANGE THE VALUES THAT ARE LOADED AND
STORED INTO THE QSPI TRANSMIT.RAM TO EXPERIMENT
WITH THE LED ARRAY AND THE QSPI.
TO RUN THE ROUTINE AGAIN, TYPE GO IN THE
DEBUG WINDOW OF EVB16.

Freescale Semiconductor, Inc.

The Periodic Interrupt Timer (INT_TEST.ASM)

The periodic interrupt timer (PIT) is an internal timer that can be programmed to make an interrupt service request at specific intervals. One application of the PIT is to configure it to interrupt the processor every second so that an interrupt service routine can update a clock.

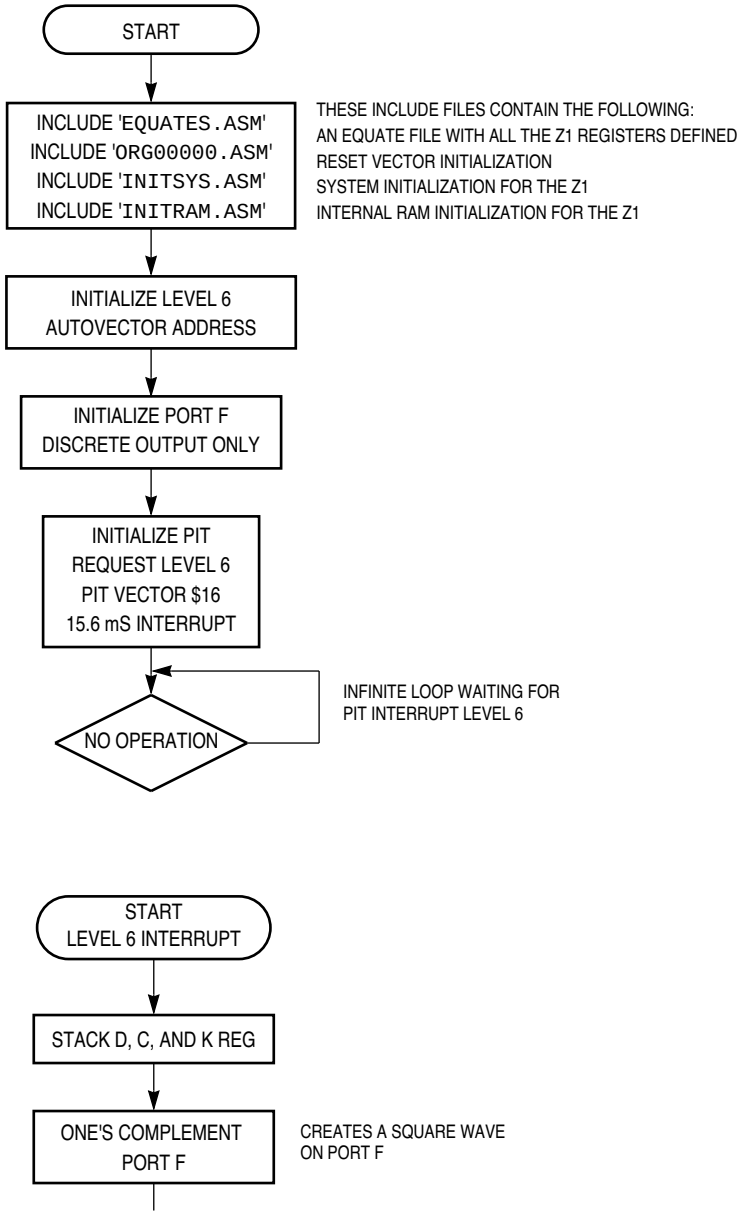
INT_TEST.ASM produces a square wave on the port F pins of the MC68HC16Z1. The square wave has a set frequency determined by the PIT timeout period. The program uses the level six autovector and the PIT times out at 15.6 ms. Port F is initialized for discrete output, then the code enters a wait loop until the programmed interval elapses. The interrupt service routine creates the square wave. **Figure 17** is a flowchart of **INT_TEST.ASM**.

For detailed information concerning interrupts, the PIT, and port F, refer to the *MC68HC16Z1 User 's Manual* (MC68HC16Z1UM/D), the *SIM Reference Manual* (SIMRM/AD), and the *CPU16 Reference Manual* (CPU16RM/AD).

INT_TEST.ASM Code Listing

```
        INCLUDE 'EQUATES.ASM'   ;table of EQUates for common register addr
        INCLUDE 'ORG00000.ASM'  ;initialize reset vector

        ORG     $0200           ;start program after interrupt vectors

*****  Initialization Routines  *****

        INCLUDE 'INITSYS.ASM'   ;initially set EK=F, XK=0, YK=0, ZK=0
                                ;set sys clock at 16.78 MHz, disable COP
        INCLUDE 'INITRAM.ASM'   ;initialize and turn on SRAM
                                ;set stack (SK=1, SP=03FE)

*****   Initialize level 6 autovector address
        LDAB    #$00
        TBEK                    ;ek extension pointer = bank0
        LDD     #INT_RT         ;load Dacc with interrupt vector addr
        STD     $002C           ;store addr to level 6 autovector

*****   Initialize  PortF       *****
        LDAB    #$0F
        TBEK                    ;ek extension pointer = bankf
        LDAB    #$00
        STAB    PFPAR           ;define port f as discrete i/o
        LDAA    #$FF
        STAA    DDRF            ;define port f as all output
        STAA    PORTF0          ;store $ff to port f

*****   Initialize the PIT      *****
        LDD     #$0616
        STD     PICR            ;pirql=6, piv=$16
```

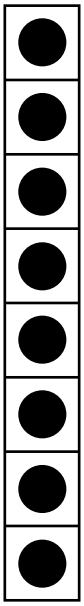
Freescale Semiconductor, Inc.



Freescale Semiconductor, Inc.

Signal Peak Detector (PEAK.ASM)

The signal peak detector graphically measures and displays the peak amplitude of a signal in real time. An audio signal is sampled at 24.95 kHz. The peak amplitude of the signal is detected, then a value that represents the peak on a bar of eight light-emitting diodes (LED) is generated. A reference value of 0.775 Vrms equivalent to 0 dB is used to relate the digital peak value to the LED display. The LED bar can display a signal in the range –15 dB to +6 dB, in 3 dB steps. **Figure 18** shows relationships between the LED bar, decibels, Vrms, and Vp. **Figure 19** shows the relationship between an analog input signal and the peak values displayed. **Figure 20** is a flowchart of **PEAK.ASM**.

LED BAR	dB	VRMS	VPEAK
	+6	1.548	2.187
	+3	1.096	1.548
	0	0.775	1.096
	-3	0.549	0.775
	-6	0.389	0.549
	-9	0.275	0.389
	-12	0.195	0.275
	-15	0.138	0.195

$$dB = 20 \bullet \log\left(\frac{Vin}{Vref}\right)$$

$$0dB \geq Vref = 0.775Vrms$$

$$Vpeak = \sqrt{2} \bullet Vrms$$

AN1233 F18

Freescale Semiconductor, Inc.

The code originating in the internal RAM will now be correctly loaded into the MC68HC16Z1.

PEAK.ASM reads values from a look-up table in memory. The file **OUTVAL2.ASM** contains the table. Be sure this file is in the same directory as **PEAK.ASM** before assembly.

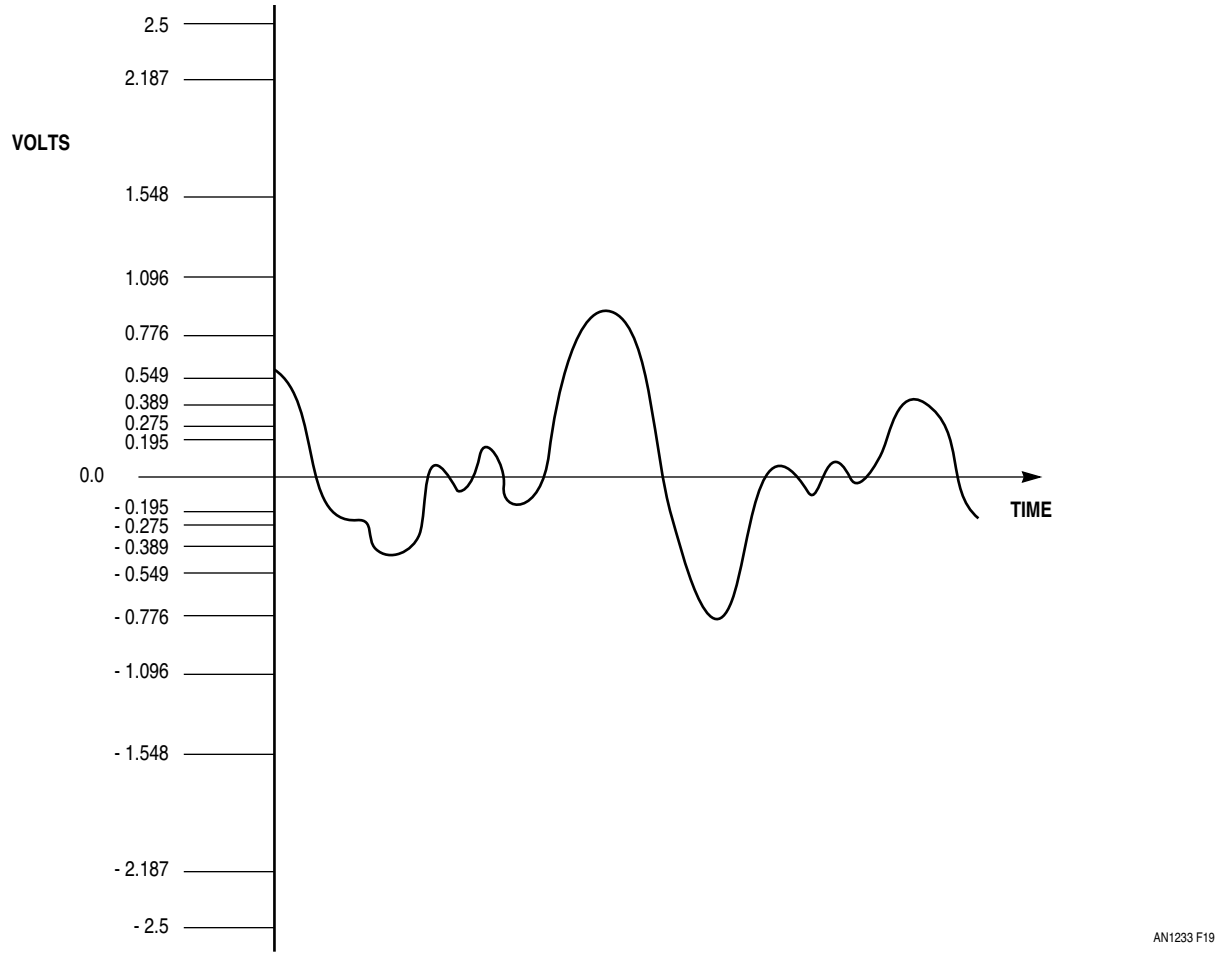


Figure 19 Analog Input vs Peak Display Level

After initializing the SRAM, the ADC, the QSPI, and the PIT, the code jumps to internal RAM at location

Freescale Semiconductor, Inc.

Using a PIT interrupt to decrement the peak value causes the LED display to decrease slowly, like a capacitor discharging, when the input signal decreases rapidly. This gives the display a more fluid appearance when rapidly-changing peak values are measured. If the display jumped from peak to peak, the discontinuity would lower the aesthetic appeal. In fact, most commercial audio analyzers show the relative peak differences of the frequency spectrum rather than attempt to display the peak signal precisely.

To test the code, hook up the system as shown in **Figure 11**. Input a known signal and observe the display. Apply an audio signal from the sound source and watch the peak detector execute in real time. If there is only one sound source output, connect it to either the left or right AFA input. The display is calibrated to the output of a CD player. The CD player puts out a line level signal, with .775 Vrms equal to 0 dB. If the sound source is not a CD player, adjust the output of the sound source so that the dynamic range of the signal is fully displayed.

PEAK.ASM Code Listing

```
        INCLUDE 'EQUATES.ASM'   ;table of EQUates for common register addr
        INCLUDE 'ORG00000.ASM'  ;initialize reset vector

*****   Temporary variable storage
PK      EQU     $0200           ; bank F
CNT     EQU     $0201           ; bank F

        ORG     $0200

*****   Initialization Routines *****

        INCLUDE 'INITSYS.ASM'   ;initially set EK=F, XK=0, YK=0, ZK=0
                                ;set sys clock at 16.78 MHz, disable COP

*****   RAM and Stack Initialization
        LDD     #$00FF
        STD     RAMBAH          ;store high ram array, bank F
        LDD     #$0000
        STD     RAMBAL          ;store low ram array, 0000
        CLR     RAMMCR          ;enable ram
        LDAB    #$0F
        TBSK                    ;set SK to bank F for system stack
        LDS     #$02FE          ;put SP in 1k internal SRAM

*****   Initialize level 6 autovector address
        LDAB    #$00
        TBEK                    ;ek extension pointer = bank0
        LDD     #JMPINT         ;load Dacc with interrupt vector addr
        STD     $002C           ;store addr to level 6 autovector

*****   Initialize the PIT      *****
        LDAB    #$0F
        TBEK                    ;ek extension pointer = bankf
        LDD     #$0616
```

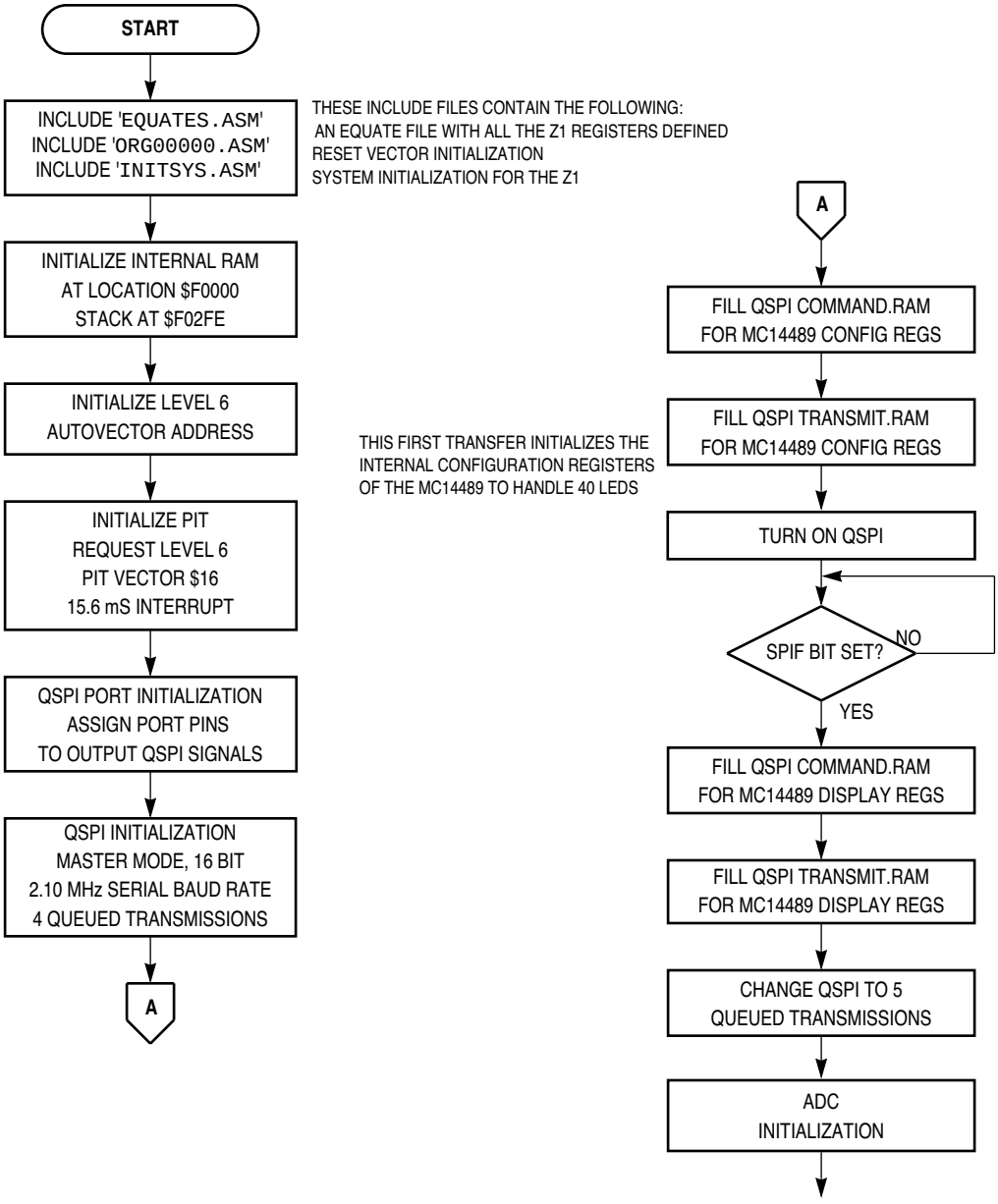
Freescale Semiconductor, Inc.

```
        STAA    CR1
        STAA    CR2
        LDAA    #$40
        STAA    CR3             ;cont=0, bitse=1, pcs0=0, no delays needed

*****   Fill QSPI Transmit.ram to write the config registers of the 14489
        LDAA    #$3F
        STD     TR0+1           ;store $3F to tran.ram registers
        STD     TR2
        STD     TR3+1
*****   Turn on the QSPI, this will write to the config registers
*****   of the MC14489 drivers
GO      LDD     #$8404
        STAA    SPCR1           ;turn on spi
SPIWT   LDAA    SPSR            ;after sending data we wait until the
        ANDA    #$80            ;spif bit is set, before we can send more
        CMPA    #$80            ;check for spi done
        BNE     SPIWT

*****   Fill QSPI Command.ram to write the display registers of the 14489
        LDAA    #$C0
        STAA    CR0             ;cont=1, bitse=1, pcs0=0, no delays needed
        STAA    CR1
        LDAA    #$40            ;cont=0, bitse=1, pcs0=0, no delays needed
        STAA    CR2
        STAA    CR4
        LDAA    #$80            ;cont=1, bitse=0, pcs0=0, no delays needed
        STAA    CR3

*****   Fill QSPI Transmit.ram for display registers of the 14489
*****   The beginning LED values will be $00, all of the LEDs will be off
        LDD     #$8000
        STD     TR0             ;TR0 = $8000
        STAA    TR3+1           ;TR1 = $0080
        LDD     #$0080          ;TR2 = $0000
        STD     TR1             ;TR3 = $XX80
        CLRD                    ;TR4 = $0000
        STD     TR2
        STD     TR4

        LDD     #$0400          ;display registers need 5 transmissions
        STD     SPCR2           ;newqp=0, endqp=4

*****   ADC Initialization      *****
        LDD     #$0000
        STD     ADCMCR          ;turn on ADC
        LDD     #$0003
        STD     ADCTL0          ;8-bit, set sample period

*****   Initialize the extension registers for the internal ram in bank F
*****   Set up the extension registers to point to bank F
        LDAB    #$0F            ;load b with $0F
        TBEK                    ;transfer Bacc to Ek
        TBXK                    ;transfer Bacc to Xk
```

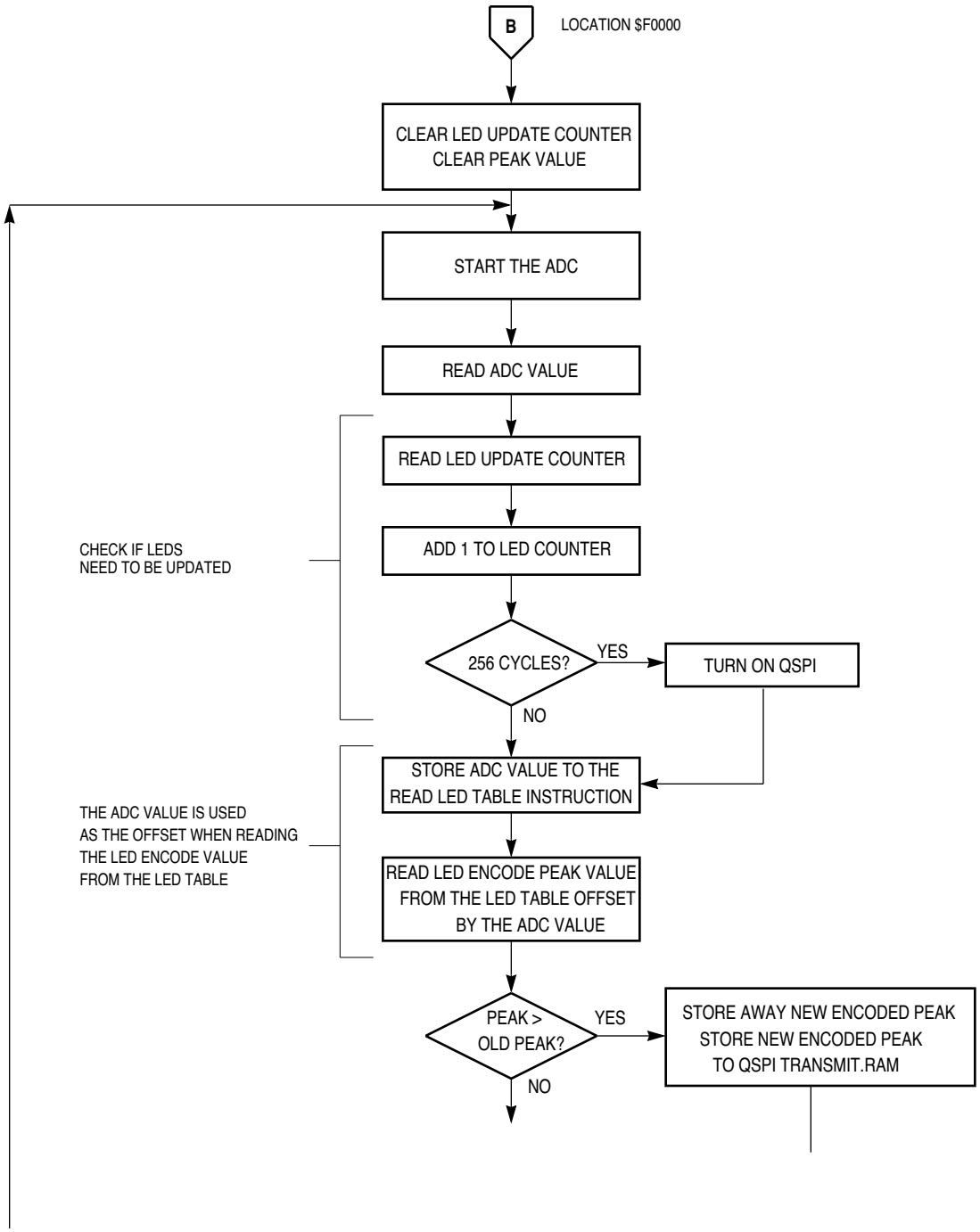
Freescale Semiconductor, Inc.

```
        LDD     #$8404          ; 6 load up d
        STD     SPCR1           ; 6 turn on QSPI, send LED data out

*       Get LED encode value from look-up table
TRAN    TED                     ; 2 transfer Eacc to Dacc
        STAA    LD+3            ; 6 Dacc high byte -> instruction ldaa $03??
        NOP                     ; 2 no operation, wait for CPU pipeline
        NOP                     ; 2 no operation, wait for CPU pipeline
LD      LDAA    LED_TBL         ; 6 load Aacc with the encoded LED value
                                ;   from scaled peak LED table

*       Update peak value if needed
        CMPA    PK              ; 6 compare value to previous peak value
        BLS     DN              ; 6,2 branch if not more than peak value
        STAA    PK              ; 6 store new peak value
        STAA    TR1             ; 6 store new value to all 5 qspi tran.rams
        STAA    TR2             ; 6
        STAA    TR2+1           ; 6
        STAA    TR4             ; 6
        STAA    TR4+1           ; 6

*****   Loop to generate calculated delay
*****   Clocks = 6 + 8*(N-1)  N >= 1
*****   N is the number put into the B accumulator

DN      LDAB    #$4B            ;  75dec this loop will create an extra delay
WAIT    DECB                    ;  to make a 24.95kHz sampling rate
        BNE     WAIT            ;  or a 668 cycle sampling period
                                ; 598 cycles

        JMP     LP              ; 6 jump back to start another conversion


*****   Exceptions/Interrupts   *****
*****   This interrupt is used to decrement each LED bar value
*****   representing the peak value of the audio signal
INT_RT  PSHM    D,CCR           ;stack Dacc and CCR on stack
        LDAA    PK              ;load Aacc with peak value
        BEQ     DONE            ;equal to 0?, then done

        ANDP    #$FEFF          ;clear C bit
        RORA                    ;rotate right once, decrease peak value
        STAA    TR1             ;store Aacc to all qspi tran.ram
        STAA    TR2
        STAA    TR2+1
        STAA    TR4
        STAA    TR4+1
        STAA    PK              ;store Aacc to peak value
        LDD     #$8404          ;load up Dacc
        STD     SPCR1           ;turn on QSPI, send LED data out

DONE    PULM    D,CCR           ;pull Dacc and CCR from stack
        RTI                     ;return from interrupt
```

Freescale Semiconductor, Inc.



Freescale Semiconductor, Inc.



Freescale Semiconductor, Inc.

A 1-kHz Bandpass Filter (1K_FLTR.ASM)

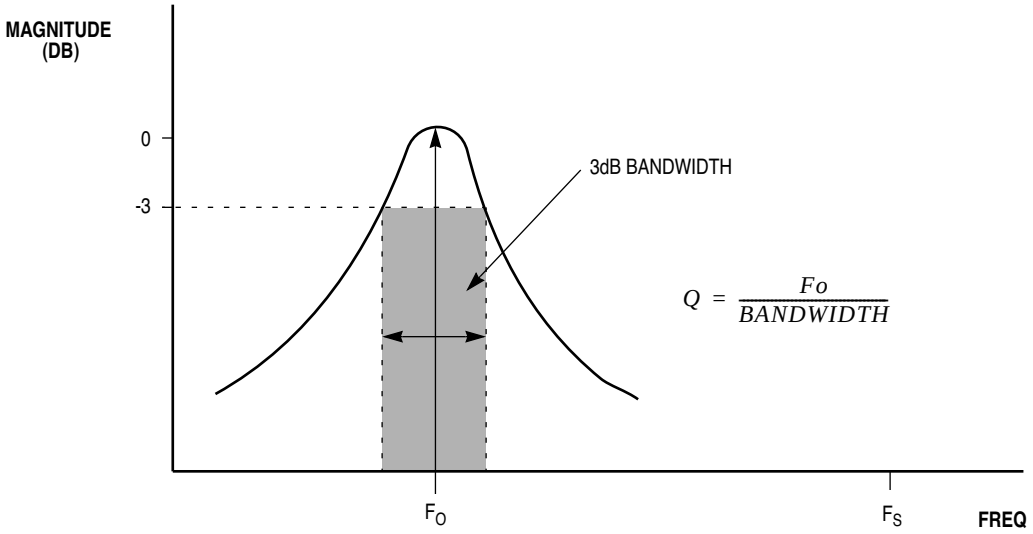
This code is similar in function to the peak detector, except that it executes a 1-kHz IIR bandpass filter on the input signal. The peak is detected and displayed on an LED bar in real time. The focus is on using the MC68HC16Z1 to implement the digital filter. **Figure 22** is a flowchart of **1K_FLTR.ASM**.

The objective is to take incoming sampled data x(n), and run the bandpass filter function on the sample to produce output y(n). Again, this is the basic 'black box' concept of electrical engineering — excite the input and watch the output change. The function in the 'black box' is defined below.

$$y(n) = 2 * \{\alpha * [x(n) - x(n-2)] + \gamma * y(n-1) - \beta * y(n-2)\}$$

This function implements an IIR bandpass function with characteristics defined by the coefficients α, β, and γ. In an RLC bandpass filter circuit, resistors, capacitors, and inductors would characterize filter response. In the digital implementation of the filter, the α, β, and γ coefficients determine the response in much the same way.

The basic parameters that define digital filter response are the Q, the sampling frequency (Fs), and the center frequency (Fo). The Q value defines the sharpness of the filter and is equal to the center frequency divided by the bandwidth between the 3 dB points. The specified sampling frequency is 24.95 kHz, the center frequency is 1 kHz, and Q value is 1.5. **Figure 21** illustrates these relationships. **Table 3** shows the way in which coefficients are stored in memory.



Freescale Semiconductor, Inc.

Equations that define the coefficients are shown below. Coefficient values are also given in the code listing.

$$\theta = \{(2 * \pi * Fo) / Fs\}$$

$$X = \theta / (2 * Q)$$

$$\text{If } X > \pi / 4 \text{ then } X = 0.75398$$

$$\beta = 0.5 * \{1 - \tan(X)\} / \{1 + \tan(X)\}$$

$$\gamma = (0.5 + \beta) * \cos\theta$$

$$\alpha = (0.5 - \beta) / 2$$

Where:

Fo = 1 kHz
Fs = 24.95 kHz
Q = 1.5

For more information concerning these equations, refer to Motorola Application Note *Digital Stereo 10-Band Generator* (APR2/D).

Once coefficient values have been obtained, they must be encoded. The assembler does not understand fractional decimal numbers, so fractional values are converted into signed 16-bit hexadecimal values. When using two's complement arithmetic, the most significant bit (bit 15) is the sign bit, and the fraction is contained in bits 14 to 0. Fifteen bits can represent the decimal numbers from 0 to 32,767. Multiply the decimal fraction by 32,768, then convert the value to the hexadecimal equivalent. Make certain that hexadecimal equivalents of negative values are in two's complement form. An example is given below.

Decimal fraction = 0.5

Multiply fractional decimal value by 32,768

$0.5 * 32{,}768 = 16{,}384$

Change decimal value to hexadecimal and binary values

16,384 dec = 4000 hex = 0100 0000 0000 0000 bin

4000 hex is the 16-bit fractional value.

CPU16 multiply and add instructions are used to implement the function. Processing is streamlined so that, in the final AFA design, five filters can be implemented in the 40.08 µs sampling period. For a more thorough discussion of the DSP instruction set and related CPU16 architecture, please consult Chapter 11 in the *CPU16 Reference Manual* (CPU16RM/AD). The processing sequence is as follows.

The ADC value x(n) is divided by two to prevent overflow.

Freescale Semiconductor, Inc.

As mentioned earlier, the 1-kHz bandpass filter is very similar to the peak detector design. Once the DSP is finished on the input x(n) sample, the peak detect algorithm is executed.

The include file **OUTVAL1.ASM** is used to encode the DSP output with an LED display value multiplied by two. Be sure this file is in the same directory as 1K_FLTR.ASM during assembly.

The best way to test this program is to connect a signal generator with sine-wave sweep capability to the AFA inputs, then set it to sweep from 0 to 15 kHz. The 1-kHz LED bar should display the amplitude of a pure 1-kHz tone and the routine should filter out higher and lower frequency signals. Since Q is equal to 1.5, some side-lobe frequencies in the pass band should be evident. For instance, if a 2-kHz pure signal is sent into the filter, the side-lobe response of the 1-kHz bandpass will pass an attenuated level of the 2-kHz tone.

1K_FLTR.ASM Code Listing

```
        INCLUDE 'EQUATES.ASM'   ;table of EQUates for common register addr
        INCLUDE 'ORG00000.ASM'  ;initialize reset vector

*****   Addresses of coefficients for the IIR Filters and initialization
COEFBS  EQU     $0280           ;base addr of coefficients
GAM_1K  EQU     COEFBS+$0       ;addr of the gamma coef
BETA_1K EQU     COEFBS+$2       ;addr of the beta coef
ALPH_1K EQU     COEFBS+$4       ;addr of the alpha coef
        ORG     $F0280
        dc.w    $7257           ;1k Hz gamma coef, Q=1.5
        dc.w    $C9F0           ;1k Hz beta coef, Q=1.5
        dc.w    $04F7           ;1k Hz alpha coef, Q=1.5

*****   Addresses of filter terms for the x(n) terms and initialization
XTRMBS  EQU     $02A0           ;base addr of x(n) filter terms
XN1_1K  EQU     XTRMBS+$0       ;x(n-1)
XN2_1K  EQU     XTRMBS+$2       ;x(n-2)
        ORG     $F02A0
        dc.w    $0000           ;1k Hz x(n-1)
        dc.w    $0000           ;1k Hz x(n-2)

*****   Addresses of filter terms for the y(n) terms and initialization
YTRMBS  EQU     $02C0           ;base addr of y(n) filter terms
YN1_1K  EQU     YTRMBS+$0       ;y(n-1)
YN2_1K  EQU     YTRMBS+$2       ;y(n-2)
X_2_1K  EQU     YTRMBS+$4       ;x(n) - x(n-2), stored here for mac
        ORG     $F02C0
        dc.w    $0000           ;1k y(n-1)
        dc.w    $0000           ;1k y(n-2)
        dc.w    $0000           ;1k [ x(n) - x(n-2) ]

*****   Addresses of various temporary variables and initialization
PKRES   EQU     $02E0           ;base addr of filter result storage
PK_1K   EQU     PKRES+$0        ;peak value for 1k Hz
CNT     EQU     PKRES+$1        ;count value for LED qspi update routine
```

Freescale Semiconductor, Inc.

```
*****   RAM and Stack Initialization
        LDD     #$00FF
        STD     RAMBAH          ;store high ram array, bank F
        LDD     #$0000
        STD     RAMBAL          ;store low ram array, 0000
        CLR     RAMMCR          ;enable ram
        LDAB    #$0F
        TBSK                    ;set SK to bank F for system stack
        LDS     #$02FE          ;put SP in 1k internal SRAM

*****   Initialize level 6 autovector address
        LDAB    #$00
        TBEK                    ;ek extension pointer = bank0
        LDD     #JMPINT         ;load Dacc with interrupt vector addr
        STD     $002C           ;store addr to level 6 autovector

*****   Initialize the PIT      *****
        LDAB    #$0F
        TBEK                    ;ek extension pointer = bankf
        LDD     #$0616
        STD     PICR            ;pirql=6, piv=$16
        LDD     #$0101
        STD     PITR            ;set the periodic timer at 62.5msec
        ANDP    #$FF1F          ;set interrupt priority to 000

*****   QSPI Initialization     *****
        LDAA    #$08
        STAA    QPDR            ;output pcs0/ss* to 0 when asserted
        LDAA    #$0F
        STAA    QPAR            ;assign QSM port pins to qspi module
        LDAA    #$FE
        STAA    QDDR            ;assign all QSM pins as outputs except miso

        LDD     #$8004          ;mstr, womq=cpol=cpha=0
        STD     SPCR0           ;16 bits, 2.10MHz serial baud rate
        LDD     #$0300          ;no interrupt generated, no wrap mode
        STD     SPCR2           ;newqp=0, endqp=3, queued for 4 trans

*****   Fill QSPI Command.ram to write the config registers of the 14489
        LDAA    #$C0
        STAA    CR0             ;cont=1, bitse=1, pcs0=0, no delays needed
        STAA    CR1
        STAA    CR2
        LDAA    #$40
        STAA    CR3             ;cont=0, bitse=1, pcs0=0, no delays needed

*****   Fill QSPI Transmit.ram to write the config registers of the 14489
        LDAA    #$3F
        STD     TR0+1           ;store $3F to tran.ram registers
        STD     TR2
        STD     TR3+1

*****   Turn on the QSPI, this will write to the config registers
*****   of the MC14489 drivers
```

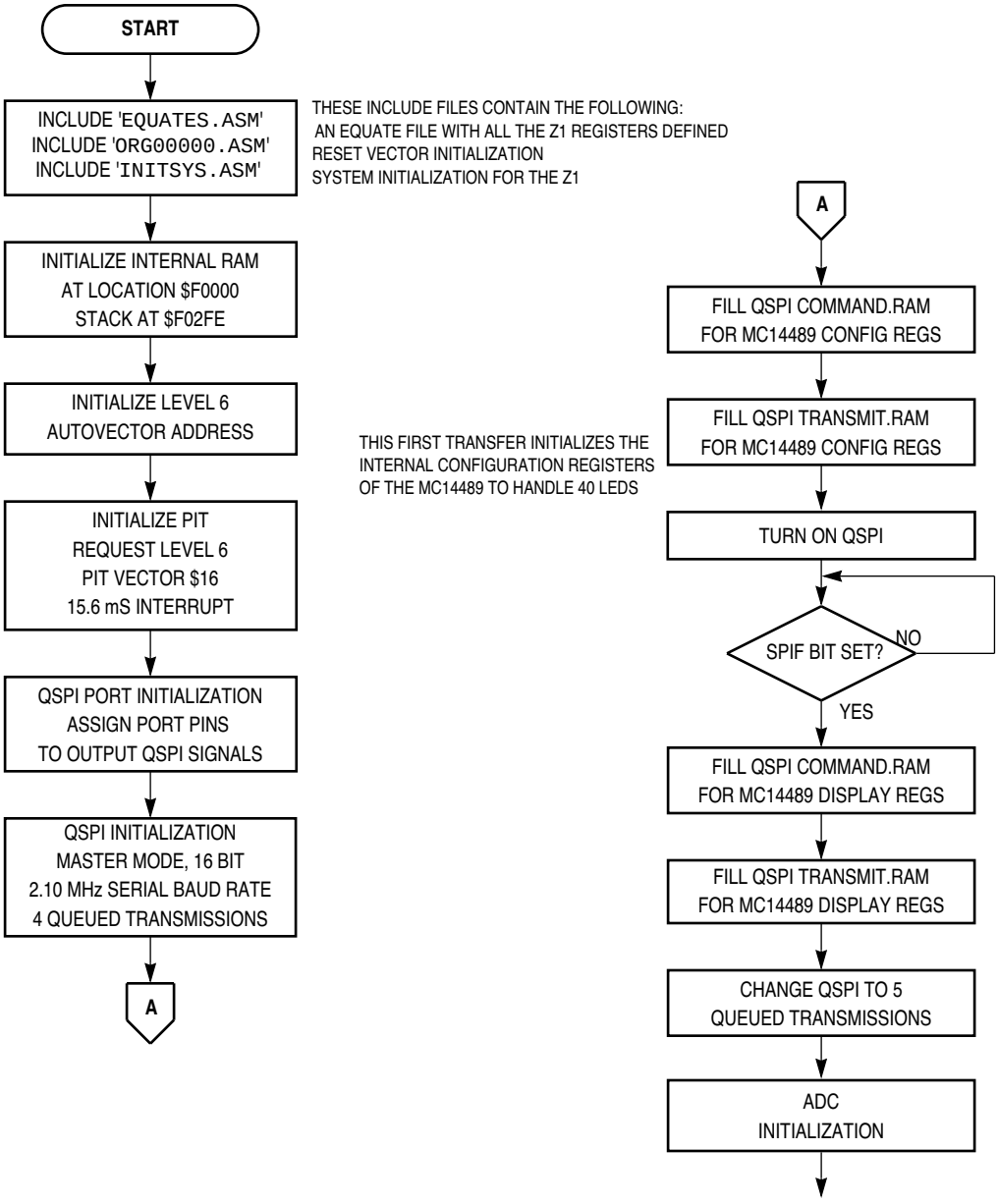
Freescale Semiconductor, Inc.

```
*****   Fill QSPI Transmit.ram for display registers of the 14489
*****   The beginning LED values will be $00, all of the LEDs will be off
        LDD     #$8000
        STD     TR0             ;TR0 = $8000
        STAA    TR3+1           ;TR1 = $0080
        LDD     #$0080          ;TR2 = $0000
        STD     TR1             ;TR3 = $XX80
        CLRD                    ;TR4 = $0000
        STD     TR2
        STD     TR4

        LDD     #$0400          ;display registers need 5 transmissions
        STD     SPCR2           ;newqp=0, endqp=4

*****   ADC Initialization      *****
        LDD     #$0000
        STD     ADCMCR          ;turn on ADC
        LDD     #$0003
        STD     ADCTL0          ;8-bit, set sample period

*****   Initialize the extension registers for the internal ram in bank F
*****   Set up the extension registers to point to bank F
        LDAB    #$0F            ;load b with $0F
        TBEK                    ;transfer Bacc to Ek
        TBXK                    ;transfer Bacc to Xk
        TBYK                    ;transfer Bacc to Yk
        TBZK                    ;transfer Bacc to Zk
        JMP     RAM             ;jump to internal ram for speed!

*****   Start of Internal 1K RAM
        ORG     $F0000
RAM     CLR     CNT             ;clear LED update counter
        CLR     PK_1K           ;clear 1K peak value

*       Initialization for DSP
        ORP     #$0010          ;set saturation mode for Macc
        CLRD                    ;clear Dacc
        TDMSK                   ;no modulo addressing
        LDY     #COEFBS         ;load y with the coef base addr
        LDX     #YTRMBS         ;load x with the yterm base addr

LP      CLRD                    ; 2 clear Dacc
        STD     ADCTL1          ; 6 single 4 conversion, single channel AD0
                                ;   writing to the ADCTL1 reg starts conv

*       Divide input x(n) by 2, no overflow problem
        LDAA    LJSRR0          ; 6 load Aacc with left jus signed ADC value
        ASRA                    ; 2 divide by 2
        STAA    AD              ; 6 store divide by 2 adc value away

*       Check if LEDs need updating
        LDAA    CNT             ; 6 load Aacc with count
        ADDA    #1              ; 2 add 1 to Aacc
        STAA    CNT             ; 6 store new count
```

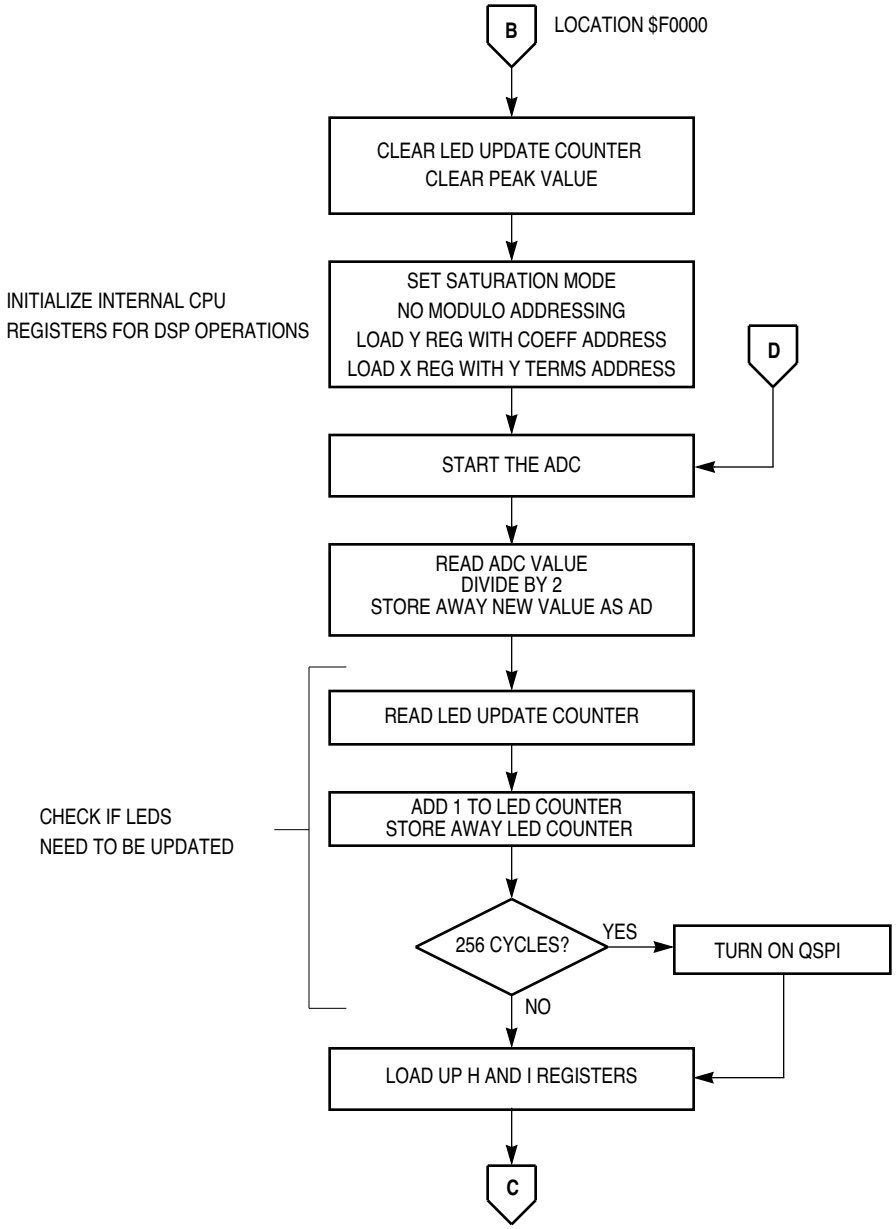
Freescale Semiconductor, Inc.

```
*       Digital processing algorithm
        TED                     ; 2 transfer Eacc to Dacc
        SUBD    XN2_1K          ; 6 Dacc = x(n) - x(n-2)
        STD     X_2_1K          ; 6 store Dacc to [x(n) - x(n-2)] addr
        LDD     XN1_1K          ; 6 load Dacc with x(n-1)
        STED    XN1_1K          ; 8 store x(n) to x(n-1) and
                                ;   store x(n-1) to x(n-2)

        MAC     2,2             ;12 gamma*(yn1)+Macc=Macc
        MAC     2,2             ;12 beta*(yn2)+Macc=Macc
        MAC     -4,-4           ;12 alpha*[x(n)-x(n-2)]+Macc=Macc
        TMET                    ; 2 transfer Macc to Eacc, truncate
        ASLE                    ; 2 multiply Eacc by 2

*       Get LED encode value from look-up table
        TED                     ; 2 transfer Eacc to Dacc
        STAA    LD1K+3          ; 6 Dacc high byte -> instruction ldaa $03??
        NOP                     ; 2 no operation, due to CPU pipeline
        NOP                     ; 2 no operation, due to CPU pipeline
LD1K    LDAA    LED_TBL         ; 6 load Aacc with the encoded LED value
                                ;   from scaled peak LED table

*       Update peak value if needed
        CMPA    PK_1K           ; 6 compare value to previous peak value
        BLS     DN1K            ; 6,2 branch if not more than peak value
        STAA    PK_1K           ; 6 store new peak value
        STAA    TR2+1           ; 6 store new value to 1k qspi tran.ram

*       Update y(n-1) and y(n-2)
DN1K    LDD     YN1_1K          ; 6 load Dacc with y(n-1)
        STED    YN1_1K          ; 8 store Eacc to y(n-1), Dacc to y(n-2)

*****   Loop to generate calculated delay
*****   Clocks = 6 + 8*(N-1)  N >= 1
*****   N is the number put into the B accumulator

        LDAB    #$3D            ;   61 this loop will create an extra delay
WAIT    DECB                    ;   to make a 24.95kHz sampling rate
        BNE     WAIT            ;   or a 668 cycle sampling period
                                ; 486 cycles
        NOP                     ; 2
        NOP                     ; 2
        NOP                     ; 2
        JMP     LP              ; 6 jump back to start another conversion


*****   Exceptions/Interrupts   *****
*****   This interrupt is used to decrement the LED bar value
*****   representing the peak value of the 1k filter band
INT_RT  PSHM    D,CCR           ;stack Dacc and CCR on stack
        LDAA    PK_1K           ;load Aacc with 1K peak value
        BEQ     DONE            ;equal to 0?, then done

        ANDP    #$FEFF          ;clear C bit
```

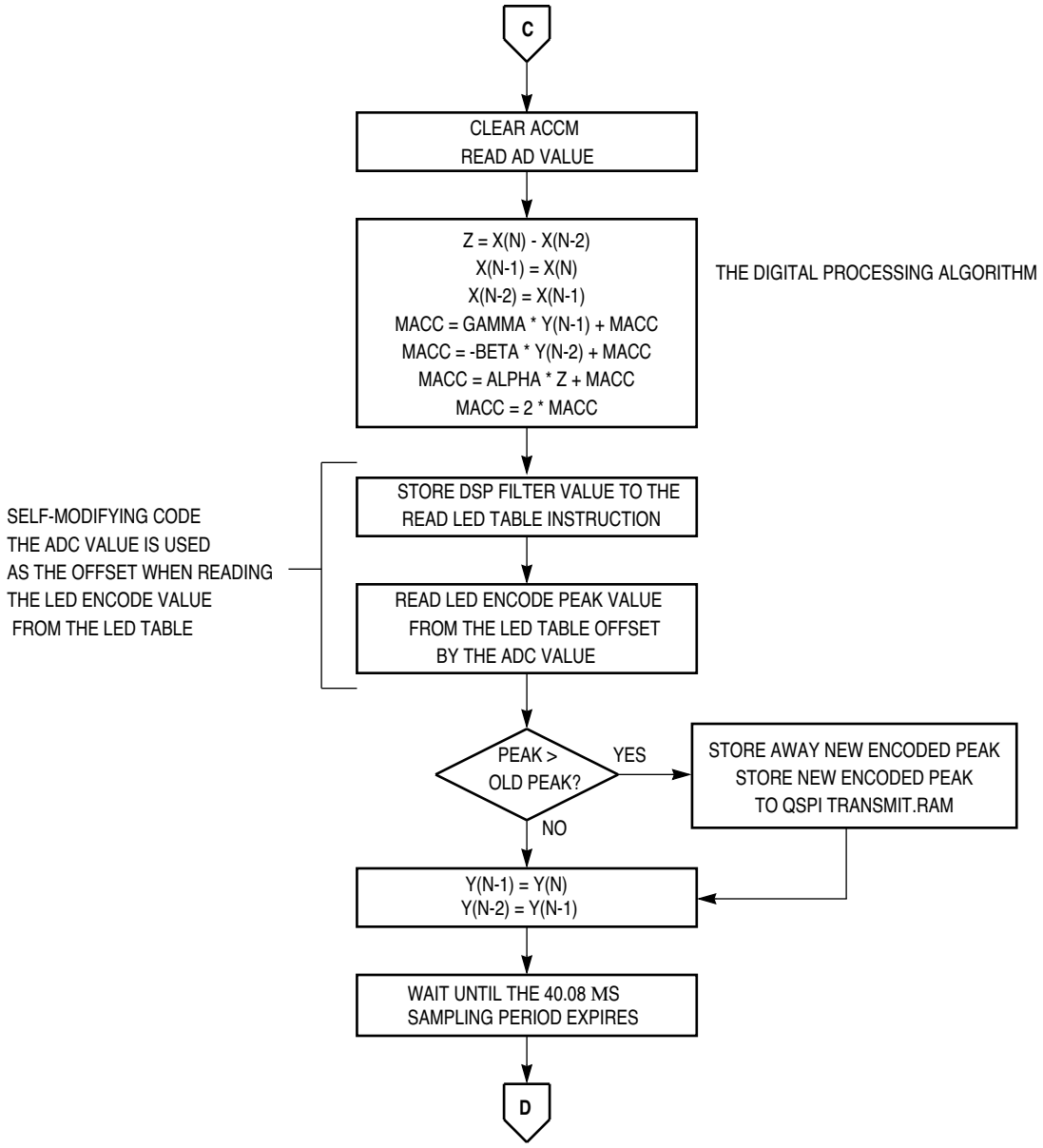
Freescale Semiconductor, Inc.



Freescale Semiconductor, Inc.



Freescale Semiconductor, Inc.



Freescale Semiconductor, Inc.

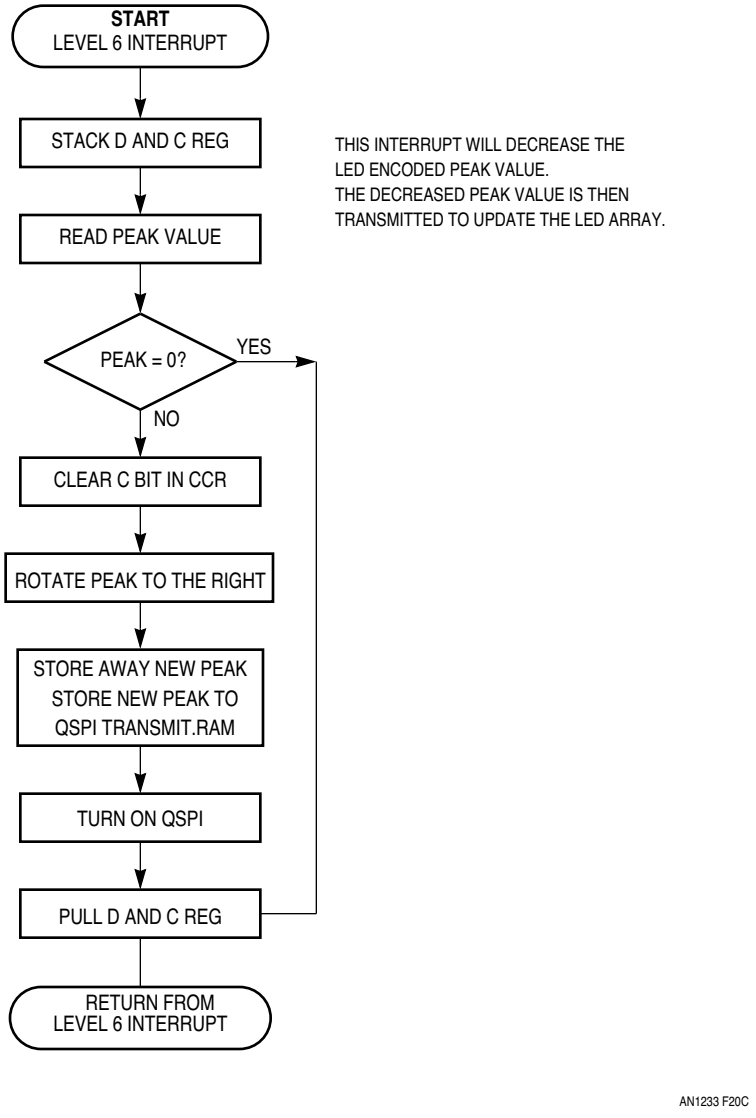


Figure 22 1K_FLTR.ASM Flowchart (Sheet 4 of 4)

Freescale Semiconductor, Inc.

The 5 Band Audio Frequency Analyzer (5BAND_SA.ASM)

The final design of the AFA is simple because of the groundwork that has already been done. **Figure 23** is a flowchart of **5BAND_SA.ASM**. Notice that five iterations of the IIR bandpass filter are executed before control passes to the interrupt routine.

The five bands and their Q values are: 125 Hz – 0.5, 500 Hz – 1.0, 1 kHz – 1.5, 4 kHz – 1.0, and 10 kHz – 0.5. Coefficient values are in the area labeled 'Address of coefficients...' at the beginning of the listing.

The specified Q values were chosen because they produce an appealing frequency display. If sharp filters with high Q values were used, the display would not show the relative differences between the bass, midrange, and treble frequency ranges. Energy associated to one particular frequency is not the primary concern of the AFA design, but rather the energy of an entire frequency band.

Test the code as before with the 1-kHz filter. Sweep a sinusoidal tone across the frequency and watch the appropriate LED array display signal energy. Apply a real time audio signal. Notice the differences between the high and low ends of the audio spectrum, the visible contrast between a bass drum and a cymbal.

5BAND_SA.ASM Code Listing

```
        INCLUDE 'EQUATES.ASM'   ;table of EQUates for common register addr
        INCLUDE 'ORG00000.ASM'  ;initialize reset vector

*****   Addresses of coefficients for the IIR Filters and initialization
COEFBS  EQU     $0280           ;base addr of coefficients
GAM_125 EQU     COEFBS+$0       ;addr of the gamma coef
BET_125 EQU     COEFBS+$2       ;addr of the beta coef
ALP_125 EQU     COEFBS+$4       ;addr of the alpha coef
GAM_500 EQU     COEFBS+$6       ;addr of the gamma coef
BET_500 EQU     COEFBS+$8       ;addr of the beta coef
ALP_500 EQU     COEFBS+$A       ;addr of the alpha coef
GAM_1K  EQU     COEFBS+$C       ;addr of the gamma coef
BET_1K  EQU     COEFBS+$E       ;addr of the beta coef
ALP_1K  EQU     COEFBS+$10      ;addr of the alpha coef
GAM_4K  EQU     COEFBS+$12      ;addr of the gamma coef
BET_4K  EQU     COEFBS+$14      ;addr of the beta coef
ALP_4K  EQU     COEFBS+$16      ;addr of the alpha coef
GAM_10K EQU     COEFBS+$18      ;addr of the gamma coef
BET_10K EQU     COEFBS+$1A      ;addr of the beta coef
ALP_10K EQU     COEFBS+$1C      ;addr of the alpha coef
        ORG     $F0280
        dc.w    $7C07           ;125 Hz gamma coef, Q=0.5
        dc.w    $C3E9           ;125 Hz beta coef, Q=0.5
        dc.w    $01F4           ;125 Hz alpha coef, Q=0.5
        dc.w    $7774           ;500 Hz gamma coef, Q=1.0
        dc.w    $C798           ;500 Hz beta coef, Q=1.0
        dc.w    $03CB           ;500 Hz alpha coef, Q=1.0
        dc.w    $7257           ;1k Hz gamma coef, Q=1.5
```

Freescale Semiconductor, Inc.

```
*****   Addresses of filter terms for the x(n) terms and initialization
XTRMBS  EQU     $02A0           ;base addr of x(n) filter terms
XN1_125 EQU     XTRMBS+$0       ;x(n-1)
XN2_125 EQU     XTRMBS+$2       ;x(n-2)
XN1_500 EQU     XTRMBS+$4       ;x(n-1)
XN2_500 EQU     XTRMBS+$6       ;x(n-2)
XN1_1K  EQU     XTRMBS+$8       ;x(n-1)
XN2_1K  EQU     XTRMBS+$A       ;x(n-2)
XN1_4K  EQU     XTRMBS+$C       ;x(n-1)
XN2_4K  EQU     XTRMBS+$E       ;x(n-2)
XN1_10K EQU     XTRMBS+$10      ;x(n-1)
XN2_10K EQU     XTRMBS+$12      ;x(n-2)
        ORG     $F02A0
        dc.w    $0000           ;125 Hz x(n-1)
        dc.w    $0000           ;125 Hz x(n-2)
        dc.w    $0000           ;500 Hz x(n-1)
        dc.w    $0000           ;500 Hz x(n-2)
        dc.w    $0000           ;1k Hz x(n-1)
        dc.w    $0000           ;1k Hz x(n-2)
        dc.w    $0000           ;1k Hz x(n-1)
        dc.w    $0000           ;1k Hz x(n-2)
        dc.w    $0000           ;1k Hz x(n-1)
        dc.w    $0000           ;1k Hz x(n-2)

*****   Addresses of filter terms for the y(n) terms and initialization
YTRMBS  EQU     $02C0           ;base addr of y(n) filter terms
YN1_125 EQU     YTRMBS+$0       ;y(n-1)
YN2_125 EQU     YTRMBS+$2       ;y(n-2)
X_2_125 EQU     YTRMBS+$4       ;x(n) - x(n-2), stored here for mac
YN1_500 EQU     YTRMBS+$6       ;y(n-1)
YN2_500 EQU     YTRMBS+$8       ;y(n-2)
X_2_500 EQU     YTRMBS+$A       ;x(n) - x(n-2), stored here for mac
YN1_1K  EQU     YTRMBS+$C       ;y(n-1)
YN2_1K  EQU     YTRMBS+$E       ;y(n-2)
X_2_1K  EQU     YTRMBS+$10      ;x(n) - x(n-2), stored here for mac
YN1_4K  EQU     YTRMBS+$12      ;y(n-1)
YN2_4K  EQU     YTRMBS+$14      ;y(n-2)
X_2_4K  EQU     YTRMBS+$16      ;x(n) - x(n-2), stored here for mac
YN1_10K EQU     YTRMBS+$18      ;y(n-1)
YN2_10K EQU     YTRMBS+$1A      ;y(n-2)
X_2_10K EQU     YTRMBS+$1C      ;x(n) - x(n-2), stored here for mac
        ORG     $F02C0
        dc.w    $0000           ;125 Hz y(n-1)
        dc.w    $0000           ;125 Hz y(n-2)
        dc.w    $0000           ;125 Hz [ x(n) - x(n-2) ]
        dc.w    $0000           ;500 Hz y(n-1)
        dc.w    $0000           ;500 Hz y(n-2)
        dc.w    $0000           ;500 Hz [ x(n) - x(n-2) ]
        dc.w    $0000           ;1k Hz y(n-1)
        dc.w    $0000           ;1k Hz y(n-2)
        dc.w    $0000           ;1k Hz [ x(n) - x(n-2) ]
        dc.w    $0000           ;4k Hz y(n-1)
        dc.w    $0000           ;4k Hz y(n-2)
        dc.w    $0000           ;4k Hz [ x(n) - x(n-2) ]
```

Freescale Semiconductor, Inc.

```
*****   Initialization Routines *****

        INCLUDE 'INITSYS.ASM'   ;initially set EK=F, XK=0, YK=0, ZK=0
                                ;set sys clock at 16.78 MHz, disable COP

*****   RAM and Stack Initialization       *
        LDD     #$00FF
        STD     RAMBAH          ;store high ram array, bank F
        LDD     #$0000
        STD     RAMBAL          ;store low ram array, 0000
        CLR     RAMMCR          ;enable ram
        LDAB    #$0F
        TBSK                    ;set SK to bank F for system stack
        LDS     #$02FE          ;put SP in 1k internal SRAM

*****   Initialize level 6 autovector address
        LDAB    #$00
        TBEK                    ;ek extension pointer = bank0
        LDD     #JMPINT         ;load Dacc with interrupt vector addr
        STD     $002C           ;store addr to level 6 autovector

*****   Initialize the PIT      *****
        LDAB    #$0F
        TBEK                    ;ek extension pointer = bankf
        LDD     #$0616
        STD     PICR            ;pirql=6, piv=$16
        LDD     #$0101
        STD     PITR            ;set the periodic timer at 62.5msec
        ANDP    #$FF1F          ;set interrupt priority to 000

*****   QSPI Initialization     *****
        LDAA    #$08
        STAA    QPDR            ;output pcs0/ss* to 0 when asserted
        LDAA    #$0F
        STAA    QPAR            ;assign QSM port pins to qspi module
        LDAA    #$FE
        STAA    QDDR            ;assign all QSM pins as outputs except miso

        LDD     #$8004          ;mstr, womq=cpol=cpha=0
        STD     SPCR0           ;16 bits, 2.10MHz serial baud rate
        LDD     #$0300          ;no interrupt generated, no wrap mode
        STD     SPCR2           ;newqp=0, endqp=3, queued for 4 trans

*****   Fill QSPI Command.ram to write the config registers of the 14489
        LDAA    #$C0
        STAA    CR0             ;cont=1, bitse=1, pcs0=0, no delays needed
        STAA    CR1
        STAA    CR2
        LDAA    #$40
        STAA    CR3             ;cont=0, bitse=1, pcs0=0, no delays needed

*****   Fill QSPI Transmit.ram to write the config registers of the 14489
        LDAA    #$3F
        STD     TR0+1           ;store $3F to tran.ram registers
```

Freescale Semiconductor, Inc.

```
*****   Fill QSPI Transmit.ram for display registers of the 14489
*****   The beginning LED values will be $00, all of the LEDs will be off
        LDD     #$8000
        STD     TR0             ;TR0 = $8000
        STAA    TR3+1           ;TR1 = $0080
        LDD     #$0080          ;TR2 = $0000
        STD     TR1             ;TR3 = $XX80
        CLRD                    ;TR4 = $0000
        STD     TR2
        STD     TR4

        LDD     #$0400          ;display registers need 5 transmissions
        STD     SPCR2           ;newqp=0, endqp=4

*****   ADC Initialization      *****
        LDD     #$0000
        STD     ADCMCR          ;turn on ADC
        LDD     #$0003
        STD     ADCTL0          ;8-bit, set sample period

*****   Initialize the extension registers for the internal ram in bank F
*****   Set up the extension registers to point to bank F
        LDAB    #$0F            ;load b with $0F
        TBEK                    ;transfer Bacc to Ek
        TBXK                    ;transfer Bacc to Xk
        TBYK                    ;transfer Bacc to Yk
        TBZK                    ;transfer Bacc to Zk
        JMP     RAM             ;jump to internal ram for speed!

*****   Start of Internal 1K RAM
        ORG     $F0000
RAM     CLR     CNT             ;clear LED update counter
        CLR     PK_125          ;clear 125 peak value
        CLR     PK_500          ;clear 500 peak value
        CLR     PK_1K           ;clear 1k peak value
        CLR     PK_4K           ;clear 4k peak value
        CLR     PK_10K          ;clear 10k peak value
        CLRW    AD              ;clear AD

*       Initialization for DSP
        ORP     #$0010          ;set saturation mode for Macc
        CLRD                    ;clear Dacc
        TDMSK                   ;no modulo addressing

LP      LDY     #COEFBS         ; 4 load y with the coef base addr
        LDX     #YTRMBS         ; 4 load x with the yterm base addr
        LDHI                    ; 8 load h and i multiplier and multiplicand
        CLRD                    ; 2 clear Dacc
        STD     ADCTL1          ; 6 single 4 conversion, single channel AD0
                                ;   writing to the ADCTL1 reg starts conv

*       Divide input x(n) by 2, no overflow problem
                LDAA   LJSRR0       ; 6 load Aacc with left jus signed ADC value
                ASRA                    ; 2 divide by 2
```

Freescale Semiconductor, Inc.

```
*       Digital processing algorithm
        TED                     ; 2 transfer Eacc to Dacc
        SUBD    XN2_125         ; 6 Dacc =  x(n) - x(n-2)
        STD     X_2_125         ; 6 store Dacc to [x(n) - x(n-2)] addr
        LDD     XN1_125         ; 6 load Dacc with x(n-1)
        STED    XN1_125         ; 8 store x(n) to x(n-1) and
                                ;   store x(n-1) to x(n-2)

        MAC     2,2             ;12 gamma*(yn1)+Macc=Macc
        MAC     2,2             ;12 beta*(yn2)+Macc=Macc
        MAC     2,2             ;12 alpha*[x(n)-x(n-2)]+Macc=Macc
        TMER                    ; 6 transfer Macc to Eacc, round for converg
        ASLE                    ; 2 multiply Eacc by 2

*       Get LED encode value from look-up table
        TED                     ; 2 transfer Eacc to Dacc
        STAA    LD125+3         ; 6 Dacc high byte -> instruction ldaa $03??
        NOP                     ; 2 no operation, due to CPU pipeline
        NOP                     ; 2 no operation, due to CPU pipeline
LD125   LDAA    LED_TBL         ; 6 load Aacc with the encoded LED value
                                ;   from scaled peak LED table

*       Update peak value if needed
        CMPA    PK_125          ; 6 compare value to previous peak value
        BLS     DN125           ; 6,2 branch if not more than peak value
        STAA    PK_125          ; 6 store new peak value
        STAA    TR4+1           ; 6 store new value to 125 qspi tran.ram

*       Update y(n-1) and y(n-2)
DN125   LDD     YN1_125         ; 6 load Dacc with y(n-1)
        STED    YN1_125         ; 8 store Eacc to y(n-1), Dacc to y(n-2)

*****   Start of the 500 Hz DSP routine
F500    CLRM                    ; 2 clear Macc
        LDE     AD              ; 6 load Eacc with AD

*       Digital processing algorithm
        TED                     ; 2 transfer Eacc to Dacc
        SUBD    XN2_500         ; 6 Dacc = x(n) - x(n-2)
        STD     X_2_500         ; 6 store Dacc to [x(n) - x(n-2)] addr
        LDD     XN1_500         ; 6 load Dacc with x(n-1)
        STED    XN1_500         ; 8 store x(n) to x(n-1) and
                                ;   store x(n-1) to x(n-2)
        MAC     2,2             ;12 gamma*(yn1)+Macc=Macc
        MAC     2,2             ;12 beta*(yn2)+Macc=Macc
        MAC     2,2             ;12 alpha*[x(n)-x(n-2)]+Macc=Macc
        TMET                    ; 2 transfer Macc to Eacc, truncate
        ASLE                    ; 2 multiply Eacc by 2

*       Get LED encode value from look-up table
        TED                     ; 2 transfer Eacc to Dacc
        STAA    LD500+3         ; 6 Dacc high byte -> instruction ldaa $03??
        NOP                     ; 2 no operation, due to CPU pipeline
        NOP                     ; 2 no operation, due to CPU pipeline
```

Freescale Semiconductor, Inc.

```
*       Digital processing algorithm
        TED                     ; 2 transfer Eacc to Dacc
        SUBD    XN2_1K          ; 6 Dacc = x(n) - x(n-2)
        STD     X_2_1K          ; 6 store Dacc to [x(n) - x(n-2)] addr
        LDD     XN1_1K          ; 6 load Dacc with x(n-1)
        STED    XN1_1K          ; 8 store x(n) to x(n-1) and
                                ;   store x(n-1) to x(n-2)

        MAC     2,2             ;12 gamma*(yn1)+Macc=Macc
        MAC     2,2             ;12 beta*(yn2)+Macc=Macc
        MAC     2,2             ;12 alpha*[x(n)-x(n-2)]+Macc=Macc
        TMET                    ; 2 transfer Macc to Eacc, truncate
        ASLE                    ; 2 multiply Eacc by 2

*       Get LED encode value from look-up table
        TED                     ; 2 transfer Eacc to Dacc
        STAA    LD1K+3          ; 6 Dacc high byte -> instruction ldaa $03??
        NOP                     ; 2 no operation, due to CPU pipeline
        NOP                     ; 2 no operation, due to CPU pipeline
LD1K    LDAA    LED_TBL         ; 6 load Aacc with the encoded LED value
                                ;   from scaled peak LED table

*       Update peak value if needed
        CMPA    PK_1K           ; 6 compare value to previous peak value
        BLS     DN1K            ; 6,2 branch if not more than peak value
        STAA    PK_1K           ; 6 store new peak value
        STAA    TR2+1           ; 6 store new value to 1k qspi tran.ram

*       Update y(n-1) and y(n-2)
DN1K    LDD     YN1_1K          ; 6 load Dacc with y(n-1)
        STED    YN1_1K          ; 8 store Eacc to y(n-1), Dacc to y(n-2)

*****   Start of the 4k Hz routine
F4K     CLRM                    ; 2 clear Macc
        LDE     AD              ; 6 load Eacc with AD

*       Digital processing algorithm
        TED                     ; 2 transfer Eacc to Dacc
        SUBD    XN2_4K          ; 6 Dacc = x(n) - x(n-2)
        STD     X_2_4K          ; 6 store Dacc to [x(n) - x(n-2)] addr
        LDD     XN1_4K          ; 6 load Dacc with x(n-1)
        STED    XN1_4K          ; 8 store x(n) to x(n-1) and
                                ;   store x(n-1) to x(n-2)

        MAC     2,2             ;12 gamma*(yn1)+Macc=Macc
        MAC     2,2             ;12 beta*(yn2)+Macc=Macc
        MAC     2,2             ;12 alpha*[x(n)-x(n-2)]+Macc=Macc
        TMET                    ; 2 transfer Macc to Eacc, truncate
        ASLE                    ; 2 multiply Eacc by 2

*       Get LED encode value from look-up table
        TED                     ; 2 transfer Eacc to Dacc
        STAA    LD4K+3          ; 6 Dacc high byte -> instruction ldaa $03??
        NOP                     ; 2 no operation, due to CPU pipeline
```

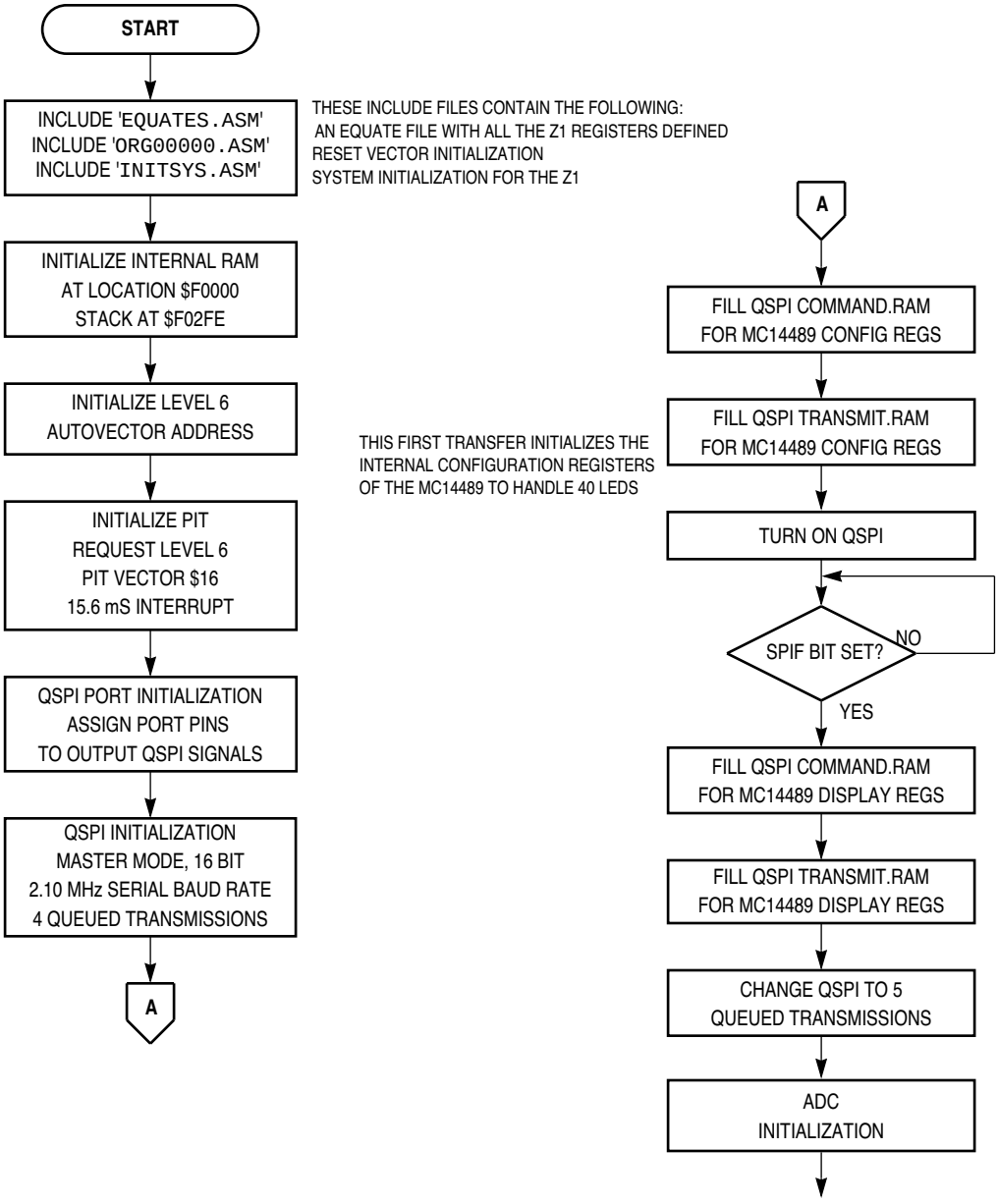
Freescale Semiconductor, Inc.

```
*       Digital processing algorithm
        TED                     ; 2 transfer Eacc to Dacc
        SUBD    XN2_10K         ; 6 Dacc =  x(n) - x(n-2)
        STD     X_2_10K         ; 6 store Dacc to [x(n) - x(n-2)] addr
        LDD     XN1_10K         ; 6 load Dacc with x(n-1)
        STED    XN1_10K         ; 8 store x(n) to x(n-1) and
                                ;   store x(n-1) to x(n-2)

        MAC     2,2             ;12 gamma*(yn1)+Macc=Macc
        MAC     2,2             ;12 beta*(yn2)+Macc=Macc
        MAC     2,2             ;12 alpha*[x(n)-x(n-2)]+Macc=Macc
        TMET                    ; 2 transfer Macc to Eacc, truncate
        ASLE                    ; 2 multiply Eacc by 2

*       Get LED encode value from look-up table
        TED                     ; 2 transfer Eacc to Dacc
        STAA    LD10K+3         ; 6 Dacc high byte -> instruction ldaa $03??
        NOP                     ; 2 no operation, due to CPU pipeline
        NOP                     ; 2 no operation, due to CPU pipeline
LD10K   LDAA    LED_TBL         ; 6 load Aacc with the encoded LED value
                                ;   from scaled peak LED table

*       Update peak value
        CMPA    PK_10K          ; 6 compare value to previous peak value
        BLS     DN10K           ; 6,2 branch if not more than peak value
        STAA    PK_10K          ; 6 store new peak value
        STAA    TR1             ; 6 store new value to 10k qspi tran.ram

*       Update y(n-1) and y(n-2)
DN10K   LDD     YN1_10K         ; 6 load Dacc with y(n-1)
        STED    YN1_10K         ; 8 store Eacc to y(n-1), Dacc to y(n-2)
        NOP

END     JMP     LP              ; 6 jump back to start another conversion
```

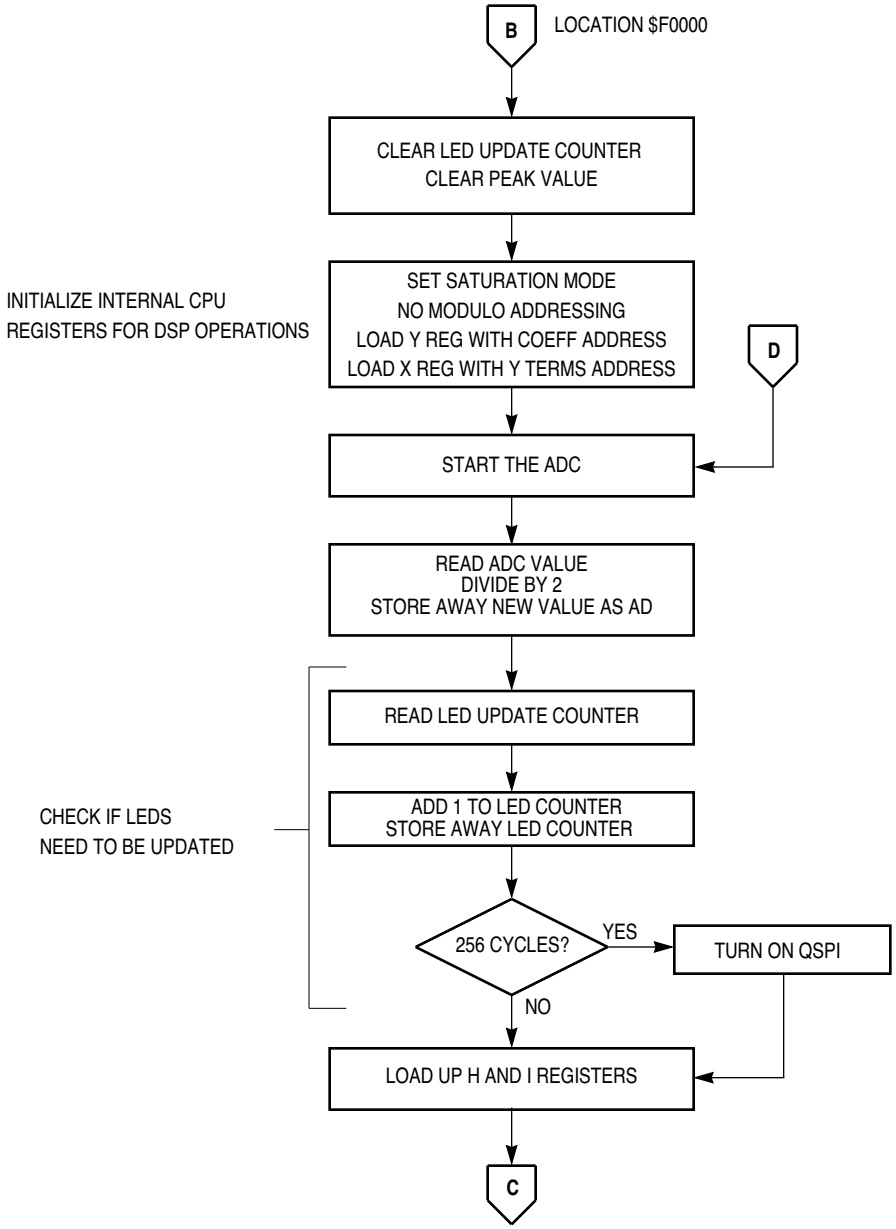
Freescale Semiconductor, Inc.

```
*****   Exceptions/Interrupts   *****
*****   This interrupt is used to decrement each LED bar value
*****   representing the peak value of each filter band
INT_RT  PSHM    D,CCR           ;stack Dacc and CCR on stack

CK125   LDAA    PK_125          ;load Aacc with 125 peak value
        BEQ     CK500           ;equal to 0?, then CK500
        ANDP    #$FEFF          ;clear C bit
        RORA                    ;rotate right once, decrease peak value
        STAA    TR4+1           ;store Aacc to 125 Hz qspi tran.ram
        STAA    PK_125          ;store Aacc to 125 Hz peak value

CK500   LDAA    PK_500          ;load Aacc with 500 peak value
        BEQ     CK1K            ;equal to 0?, then CK1K
        ANDP    #$FEFF          ;clear C bit
        RORA                    ;rotate right once, decrease peak value
        STAA    TR4             ;store Aacc to 500 Hz qspi tran.ram
        STAA    PK_500          ;store Aacc to 500 Hz peak value
CK1K    LDAA    PK_1K           ;load Aacc with 1k peak value
        BEQ     CK4K            ;equal to 0?, then CK4K
        ANDP    #$FEFF          ;clear C bit
        RORA                    ;rotate right once, decrease peak value
        STAA    TR2+1           ;store Aacc to 1k Hz qspi tran.ram
        STAA    PK_1K           ;store Aacc to 1k Hz peak value

CK4K    LDAA    PK_4K           ;load Aacc with 4k peak value
        BEQ     CK10K           ;equal to 0?, then CK10K
        ANDP    #$FEFF          ;clear C bit
        RORA                    ;rotate right once, decrease peak value
        STAA    TR2             ;store Aacc to 4k Hz qspi tran.ram
        STAA    PK_4K           ;store Aacc to 4k Hz peak value

CK10K   LDAA    PK_10K          ;load Aacc with 10k peak value
        BEQ     UPDATE          ;equal to 0?, then UPDATE
        ANDP    #$FEFF          ;clear C bit
        RORA                    ;rotate right once, decrease peak value
        STAA    TR1             ;store Aacc to 10k Hz qspi tran.ram
        STAA    PK_10K          ;store Aacc to 10k Hz peak value

UPDATE  LDD     #$8404          ;load up Dacc
        STD     SPCR1           ;turn on QSPI, send LED data out

DONE    PULM    D,CCR           ;pull Dacc and CCR from stack
        RTI                     ;return from interrupt

*****   Location of start of level 6 interrupt, has to be in bank 0
        ORG     $A000
JMPINT  JMP     INT_RT

*****   OUTVAL1 is a 256 byte lookup table to convert an
*****   ADC reading to a LED value that can be transmitted to the 14489
*****   Multiplies by two and
*****   Encodes to a scale of +6, +3, 0, -3, -6, -9, -12, -15 dB
```

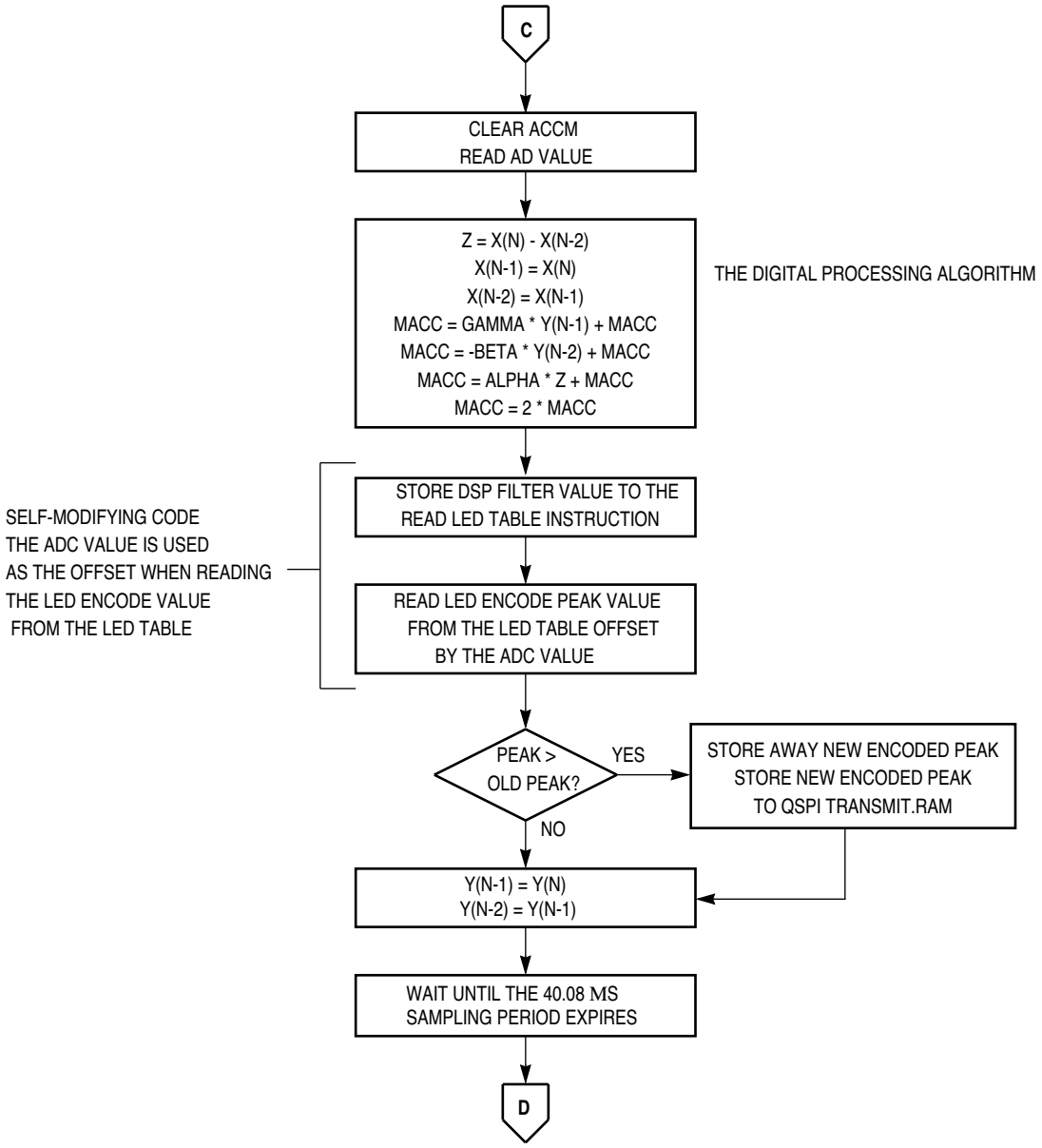
Freescale Semiconductor, Inc.



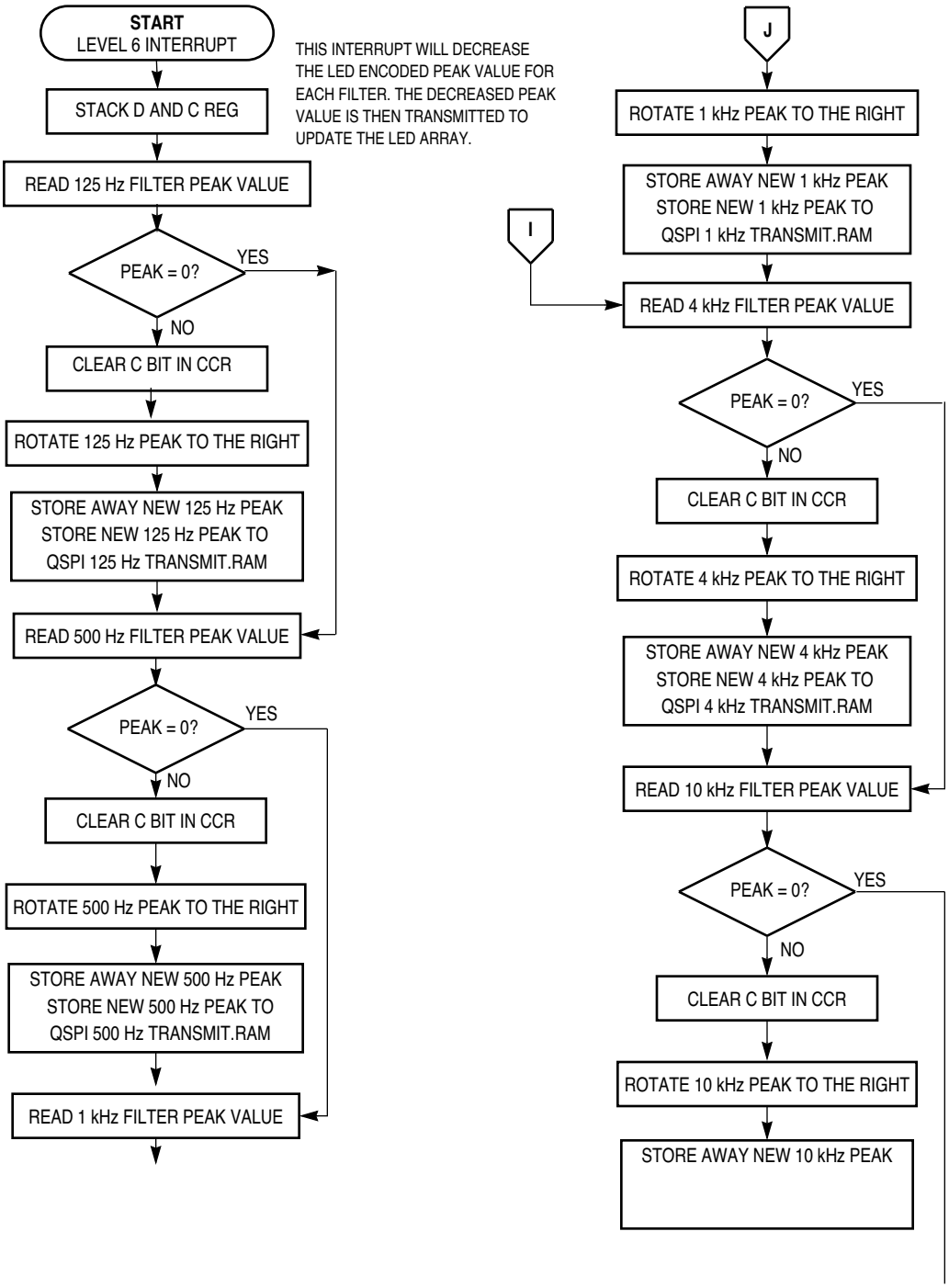
Freescale Semiconductor, Inc.



Freescale Semiconductor, Inc.



Freescale Semiconductor, Inc.


START
LEVEL 6 INTERRUPT
THIS INTERRUPT WILL DECREASE THE LED ENCODED PEAK VALUE FOR EACH FILTER. THE DECREASED PEAK VALUE IS THEN TRANSMITTED TO UPDATE THE LED ARRAY.
STACK D AND C REG
READ 125 Hz FILTER PEAK VALUE
PEAK = 0?
YES
NO
CLEAR C BIT IN CCR
ROTATE 125 Hz PEAK TO THE RIGHT
STORE AWAY NEW 125 Hz PEAK
STORE NEW 125 Hz PEAK TO
QSPI 125 Hz TRANSMIT.RAM
READ 500 Hz FILTER PEAK VALUE
PEAK = 0?
YES
NO
CLEAR C BIT IN CCR
ROTATE 500 Hz PEAK TO THE RIGHT
STORE AWAY NEW 500 Hz PEAK
STORE NEW 500 Hz PEAK TO
QSPI 500 Hz TRANSMIT.RAM
READ 1 kHz FILTER PEAK VALUE
J
ROTATE 1 kHz PEAK TO THE RIGHT
STORE AWAY NEW 1 kHz PEAK
STORE NEW 1 kHz PEAK TO
QSPI 1 kHz TRANSMIT.RAM
I
READ 4 kHz FILTER PEAK VALUE
PEAK = 0?
YES
NO
CLEAR C BIT IN CCR
ROTATE 4 kHz PEAK TO THE RIGHT
STORE AWAY NEW 4 kHz PEAK
STORE NEW 4 kHz PEAK TO
QSPI 4 kHz TRANSMIT.RAM
READ 10 kHz FILTER PEAK VALUE
PEAK = 0?
YES
NO
CLEAR C BIT IN CCR
ROTATE 10 kHz PEAK TO THE RIGHT
STORE AWAY NEW 10 kHz PEAK

Freescale Semiconductor, Inc.

CONCLUSION

This application note is intended to give designers some insight concerning the use of digital signal processing algorithms with a microcontroller. The finished project is flexible enough to permit experimenting with different filters and LED output displays. DSP allows the experimenter to make on-the-fly changes in filter response by changing the coefficients.

REFERENCES

The following Motorola documents are referred to in this application note.

- *M68HC16Z1EVB User's Manual* (M68HC16Z1EVB/D)
- *MC68HC16Z1 User's Manual* (MC68HC16Z1UM/D)
- *CPU16 Reference Manual* (CPU16RM/AD)
- *QSM Reference Manual* (QSMRM/AD)
- *ADC Reference Manual* (ADCRM/AD)
- MC14489 Data Sheet (MC14489/D)

These items can be obtained through a Motorola Sales Office or Literature Distribution Center.

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.
Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.